PROYECTO INTEGRADOR DE INGENIERÍA NUCLEAR

CÁLCULO DE BLINDAJES ASOCIADO A GUÍAS DE NEUTRONES

Osiris Inti Abbate

Mgtr. Ariel Aníbal Márquez
Director
Dr. José Ignacio Márquez Damián
Co-director

Miembros del Jurado
Dra. María Arribere (Instituto Balseiro)
Ing. Daniel Hergenreder (INVAP S.E.)

Junio de 2020

Departamento de Física de Reactores y Radiaciones
Centro Atómico Bariloche

Instituto Balseiro
Universidad Nacional de Cuyo
Comisión Nacional de Energía Atómica
Argentina

INVENTARIO 24116
21.05.21
Biblioteca Leo Falicov

Al estado, que tanto invirtió en mí

Índice de símbolos

- E: Energía.
- x, y, z: Variables de coordenadas espaciales, siendo z la dirección principal de propagación del haz, e y la dirección vertical.
- $\hat{\Omega}$: Dirección de propagación de una partícula.
- θ: Ángulo entre $\hat{\Omega}$ y la dirección de interés (ej.: z).
- $\mu = cos(\theta)$.
- ϕ: Ángulo entre el eje principal del plano normal a la dirección de interés (ej.: x) y la proyección de $\hat{\Omega}$ en dicho plano, siguiendo la regla de la mano derecha.
- H*(10): Dosis equivalente ambiental.
- m de un espejo neutrónico: $m_c = Q_c/Q_{c-Ni}$.
- Tracks: Propiedades de partículas registradas al atravesar una superficie.
- Tally: Conteo de eventos en un volumen o superficie, o conjunto de éstos.
- Bins: Intervalos en los que se divide una variable para confeccionar un histograma.
- Alocar: Reservar memoria para una variable (del inglés *allocate*).
- String: Conjunto de caracteres.
- Loop: Conjunto de código repetido varias veces.

Índice de contenidos

Índice de símbolos v

Índice de contenidos vii

Índice de figuras xi

Índice de tablas xiii

Resumen xv

Abstract xvii

1. Introducción 1

1.1. Objetivos y motivaciones . . . 1
1.2. Desarrollos de fuentes de distribuciones de Fairhurst y Ayala . . . 2
1.3. McStas . . . 4
1.4. Tripoli . . . 4
1.5. El Reactor RA10 . . . 5

2. Desarrollo de herramientas computacionales 7

2.1. Arquitectura computacional . . . 7
2.2. Primeras implementaciones de detector y fuente sobre guía . . . 8
2.3. Procesamiento de fuentes de *tracks* de Tripoli . . . 11
2.4. Biblioteca de fuentes de distribuciones en lenguaje C . . . 12
2.4.1. Conceptos generales . . . 13
2.4.2. DSource . . . 15
2.4.3. Distrib . . . 19
2.4.4. Grid . . . 23
2.5. Geometrías implementadas . . . 24
2.5.1. Geometría “window” . . . 26
2.5.2. Geometría “guide” . . . 27
2.5.3. Geometría “activ” . . . 27

2.6. Integración de las herramientas desarrolladas con los códigos McStas y Tripoli . . . 28
2.7. Biblioteca de fuentes de distribuciones en Python . . . 31

3. Cálculo de blindajes en un modelo conceptual de guía de neutrones 35
3.1. Consideraciones generales sobre los cálculos con fuentes de distribuciones 35
3.2. Descripción de los modelos implementados en McStas y Tripoli . . . 38
3.3. Esquema de la simulación . . . 44
3.4. Caracterización de las fuentes de *tracks* utilizadas . . . 46
3.5. Resultados de las simulaciones . . . 51
3.5.1. Discretizaciones empleadas . . . 51
3.5.2. Tubo de vuelo . . . 52
3.5.3. Interior de la guía . . . 53
3.5.4. Búnker de guías . . . 57
3.5.5. *Hall* de guías . . . 64
3.5.6. Resultados finales . . . 71

4. Conclusiones 75

A. Documentación de la biblioteca DSource 77
A.1. Estructuras . . . 77
A.1.1. DSource . . . 77
A.1.2. Distrib . . . 78
A.1.3. Grid . . . 79
A.2. Funciones de interfaz de usuario . . . 79
A.2.1. DS_create . . . 79
A.2.2. DS_I_tot . . . 80
A.2.3. DS_p2_tot . . . 80
A.2.4. DS_N_tot . . . 80
A.2.5. DS_transf . . . 80
A.2.6. DS_is_in . . . 81
A.2.7. DS_save . . . 81
A.2.8. DS_sort . . . 82
A.2.9. DS_save_distribs . . . 82
A.2.10. DS_read_info . . . 82
A.2.11. DS_read_tracks . . . 83
A.2.12. DS_destroy . . . 83
A.3. Funciones de geometría . . . 83
A.3.1. [geom]_transf . . . 83
A.3.2. [geom]_from_grids . . . 84

A.3.3. [geom]_from_distribs 85
A.3.4. [geom]_headers 86
A.3.5. [geom]_from_tally 86
A.3.6. [geom]_destroy 87
A.4. Funciones para gráficos en Python 87
A.4.1. Distrib.plot 87
A.4.2. Distrib.plot_I 88
A.4.3. Distrib.plot_cdf 88
A.4.4. Distrib.plot_2D 88

Bibliografía **91**

Agradecimientos **93**

Índice de figuras

1.1. Esquema de discretizaciones para fuentes planas rectangulares. 3
1.2. Pileta del RA10 5

2.1. Esquema de discretizaciones para guías. 9
2.2. Esquema de variables de parametrización de guía curva. 10
2.3. Esquema de los parámetros de `DSource`. 16
2.4. Esquema de las funciones de manejo de `DSource`. 17
2.5. Esquema de los archivos para el guardado de fuentes de distribuciones. 18
2.6. Esquema de los parámetros de `Distrib`. 19
2.7. Esquema de los códigos empleados 29
2.8. Figuras generadas con biblioteca `dsource` en Python. 33

3.1. Esquema de los tramos de la guía GF1 39
3.2. Modelo de búnker y *hall* de guías en Tripoli. 40
3.3. Dimensiones del modelo de guías. 42
3.4. Dimensiones de los elementos del *hall* de guías. 43
3.5. Dimensiones del modelo de *beam catcher*. 43
3.6. Modelo para la simulación del exterior del búnker. 46
3.7. Ubicación de la fuente de *tracks*. 47
3.8. Corriente de neutrones fríos, térmicos, epitérmicos y rápidos, en micro zonas. 48
3.9. Corriente en función de micro grupos de energía, variando μ. 48
3.10. Distribución angular de los neutrones fríos de fuente. 49
3.11. Corriente en función de micro grupos de energía, variando μ. 49
3.12. Corriente de fotones en micro zonas, para los 4 grupos de energía. . . . 50
3.13. Distribución angular de los fotones de fuente. 50
3.14. Corrientes de neutrones en la sección de entrada a la guía GF1. 52
3.15. Corrientes de fotones en la sección de entrada a la guía GF1. 53
3.16. Corriente de neutrones escapando de la guía en función de z. 55
3.17. Espectro de los neutrones escapando de la guía, para varios valores de de z. 55

3.18. Corriente de neutrones escapando de la guía, en función de z y la variable `mirror`. 56
3.19. Distribución angular de la corriente de neutrones escapando de la guía. 56
3.20. Corriente de fotones escapando de la guía en función de z. 57
3.21. Corriente de fotones escapando de la guía, en función de z y la variable `mirror`. 57
3.22. Mapas de los *tallies* de activación registrados. 58
3.23. Dosis ambiental en un corte horizontal del búnker de guías. 60
3.24. Dosis ambiental en un corte transversal del búnker de guías. 61
3.25. Comparación entre dosis empleando fotones, electrones y positrones, o sólo fotones. 62
3.26. Corriente de fotones *prompt* entrando a las paredes laterales del búnker. 62
3.27. Corriente de fotones de fuente entrando a la pared frontal del búnker. . 63
3.28. Espectro energético de fotones de fuente, de reacciones *prompt* y de activación. 63
3.29. Primer resultado de dosis en un plano horizontal en el blindaje exterior. 65
3.30. Primer resultado de dosis en un plano transversal en el blindaje exterior. 65
3.31. Perfil de dosis a lo largo del eje y, con la recta de extrapolación. 65
3.32. Segundo resultado de dosis en un plano transversal en el blindaje exterior. 66
3.33. Perfil de dosis a lo largo del eje y, con la recta de extrapolación. 66
3.34. Dosis ambiental en un corte transversal del blindaje exterior. 67
3.35. Dosis ambiental por fotones de activación en un corte transversal del blindaje exterior. 67
3.36. Dosis ambiental en un corte horizontal del búnker de guías. 69
3.37. Dosis ambiental en un corte transversal del búnker de guías. 70
3.38. Comparación entre las componentes de dosis en un corte transversal del búnker de guías. 70
3.39. Dosis total en un plano horizontal del búnker. 72
3.40. Dosis total en un plano transversal del búnker. 72
3.41. Dosis total en un plano horizontal del blindaje exterior. 73
3.42. Dosis total en un plano transversal del blindaje exterior. 73

Índice de tablas

2.1. Descripción de las variables de parametrización de guía 9
2.2. Esquema de discretizaciones para `window` 26
2.3. Esquema de discretizaciones para `guide` 27
2.4. Esquema de discretizaciones para `activ` 28

3.1. Dimensiones de los tramos de GF1. 39
3.2. Materiales usados en el modelo en Tripoli. 41
3.3. Resultados de las simulaciones del interior de la guía 54
3.4. Esquema de las simulaciones realizadas. 71

Resumen

El presente proyecto consistió en el desarrollo de una cadena de cálculo de blindajes, con aplicación particular a guías de neutrones, mediante los códigos McStas y Tripoli.

Se desarrolló una biblioteca en lenguaje C, en la cual se implementaron las funciones principales para el uso de fuentes de distribuciones. Esta herramienta permite la construcción de fuentes en base a listas de *tracks* de MCNP o Tripoli, o mediante la detección de partículas en McStas, y la posterior producción de neutrones o fotones conservando las distribuciones originales. Se implementaron las siguientes geometrías: fuente plana rectangular, fuente con forma de guía de neutrones, para modelar los escapes a través de los espejos, y fuente volumétrica, para la activación neutrónica en materiales. Las principales utilidades del desarrollo son la reducción de varianza y el acople entre códigos.

Para demostrar el funcionamiento de la herramienta, se realizó el cálculo de blindajes de un diseño conceptual para la guía GF1 de RA10. Se logró caracterizar el campo de dosis en posiciones lejanas al reactor, así como verificar el cumplimiento de los límites de dosis requeridos.

Palabras clave: MONTE CARLO, BLINDAJES, MCSTAS, TRIPOLI, RA10

Abstract

The present project consisted in the development of a shielding calculation line, with particular application on neutron guides, making use of the codes McStas and Tripoli.

A library in C languaje was developed, in which the main functions for the use of distributional sources were implemented. This tool allows the construction of sources based on tracks list of MCNP or Tripoli, or by the detection of particles in McStas, and the posterior production of neutrons or photons preserving the original distributions. The following geometries were implemented: rectangular flat source, source shaped like a neutron guide, for modelling scapes through the mirrors, and volumetric source, for neutronic activation in materials. The main utilities of the development are variance reduction and coupling between codes.

To demonstrate the tool operation, a shielding calculation on a conceptual design for the GF1 guide of RA10 was performed. It was possible to characterize the dose field in distant positions from the reactor, as well as to verify the fulfillment of the required dose limits.

Keywords: MONTE CARLO, SHIELDING, MCSTAS, TRIPOLI, RA10

Capítulo 1

Introducción

1.1. Objetivos y motivaciones

El cálculo de blindajes consiste en la obtención de valores de magnitudes dosimétricas, mediante cálculos computacionales. Es de suma importancia en instalaciones con fuentes de radiaciones ionizantes, y particularmente en reactores nucleares. El objetivo final de los cálculos es determinar si parámetros como la tasa de dosis equivalente ambiental son adecuados para la presencia de personas, y/o qué clase de restricciones deberán ser tenidas en cuenta para la correcta protección radiológica del personal expuesto. En particular, la norma AR 4.1.1 de la Autoridad Regulatoria Nuclear argentina [1] establece que en los locales sin restricciones para el acceso de trabajadores la tasa de dosis equivalente ambiental no debe superar los 3 µSv/h. Normalmente será requerido el empleo de blindajes que separen la fuente de radiación de los recintos donde se pretende alojar trabajadores, cuyo material dependerá del tipo de partículas predominantes en la dosis ambiental.

Con respecto a los métodos computacionales empleados en el cálculo de blindajes, se debe tener en cuenta la gran anisotropía de los campos de radiación al atravesar medios absorbentes, la cual dificulta la aplicación de métodos determinísticos. Por este motivo una de las técnicas más utilizadas, y a la cual se restringe este proyecto, es el método Monte Carlo. En este caso la distribución angular no se ve distorsionada por el cálculo, pero crece el costo computacional de obtener estadística apropiada en la zona de interés. En este sentido, el problema radica en que, al realizarse cálculos de blindajes, se emplea como fuente la radiación proveniente de una zona con alta tasa de dosis, y el punto de interés para el cálculo de H*(10) es la región donde se espera que ésta sea adecuada para las personas. Por lo tanto, durante las simulaciones, se debe registrar una cantidad apropiada de partículas en puntos donde justamente se busca que haya una baja presencia de partículas. Esto introduce la necesidad de técnicas de reducción de varianza, es decir métodos para propagar preferencialmente la radiación

hacia la zona de interés.

En el presente Proyecto Integrador se planteó como objetivo desarrollar una cadena de cálculo implementando la utilización de fuentes de distribuciones en guías de neutrones. Las fuentes de distribuciones consisten justamente en una técnica de reducción de varianza, ya que permiten en una simulación caracterizar la corriente en una interfaz de interés, para luego continuar la propagación sólo a través de ésta. En este caso la idea es implementar dichas fuentes en los espejos de guías de neutrones, capturando la distribución de los neutrones no reflejados, para luego introducir esta fuente de radiación en un cálculo de blindajes de la periferia de dicho instrumento, normalmente denominado *hall* de guías.

Las cadenas de cálculo como la que se pretende desarrollar parten de una lista de *tracks* registrada durante una corrida de núcleo, en la cual se caracteriza la corriente ingresando a alguno de los haces de un reactor de investigación. Las partículas registradas atravesando la "ventana" consisten tanto en neutrones como fotones, en adelante denominados partículas de fuente. Durante la simulación de la propagación de los neutrones en la periferia del reactor se producirán nuevos fotones, tanto en reacciones (n, γ) *prompt*, es decir gammas provenientes del decaimiento del núcleo excitado luego de la captura neutrónica, como en el decaimiento β de núcleos activados neutrónicamente. La cadena de cálculo implementada en el presente proyecto captura las cuatro componentes de la dosis ambiental en la periferia de la fuente de radiación: neutrones de fuente, fotones de fuente, fotones *prompt* y fotones de activación.

1.2. Desarrollos de fuentes de distribuciones de Fairhurst y Ayala

En los proyectos integradores de Roberto Fairhurst [2] y Jonathan Ayala [3] se desarrollaron técnicas para la implementación de fuentes de distribuciones planas rectangulares, para su uso en los códigos McStas y Tripoli. Su uso estaba orientado, en el caso de Fairhurst, a la propagación de neutrones por el interior de guías con McStas, para caracterizar las distribuciones a la salida de los haces de RA10. En el caso de Ayala, se introdujo el código Tripoli, y se demostró la utilidad de la cadena de cálculo en un cálculo de blindajes en un conducto de extracción también de RA10.

Cada utilización de una fuente de distribuciones consta de dos etapas, una de detección y otra de producción. En la etapa de detección, las partículas que atraviesan el plano de la fuente son clasificadas según sus variables E, x, y, μ y ϕ, donde x e y forman el plano transversal a la dirección z de propagación de los neutrones. Dicha clasificación consiste en la creación de histogramas para cada variable, basados en una estructura de discretizaciones dada, es decir que para cada partícula detectada se determina a qué

grupo o *bin* de cada histograma pertenece y se suma su peso estadístico allí. Pero debido a que la distribución en función de cada variable tiene una dependencia con el valor de las otras, es necesaria una técnica para capturar dicha correlación entre variables. Ésta consistió en la implementación de histogramas anidados, de modo que, por ejemplo, no se cuenta con una única distribución para la variable x, sino una por cada grupo de energía. Del mismo modo, las histogramas para y, μ y ϕ están anidados sucesivamente, es decir que la cantidad de distribuciones en función de ϕ es igual al producto de la cantidad de grupos de E, x, y y μ. Para conservar una estadística apropiada en todos los histogramas, y a la vez contar con curvas de probabilidad relativamente suaves en función de cada variable, se empleó un esquema de grupos gruesos y finos, denominados macro y micro grupos, o macro y micro zonas. Las grillas gruesas se emplean para las variables de correlación, mientras que las finas se usan en las distribuciones finales. En la Figura 1.1 se muestra un esquema de la estructura de discretizaciones empleada. Las variables μ y ϕ determinan la dirección de propagación $\hat{\Omega}$, de modo que:

$$\hat{\Omega} = (sin\theta cos\phi, sin\theta sin\phi, cos\theta) = (\sqrt{1-\mu^2}cos\phi, \sqrt{1-\mu^2}sin\phi, \mu) \tag{1.1}$$

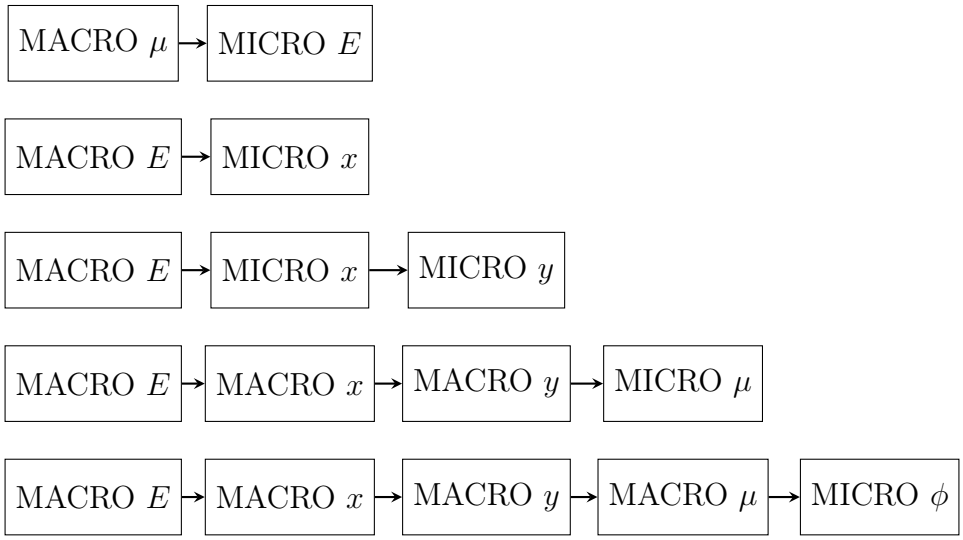


Figura 1.1: Esquema de la estructura de discretizaciones empleada para fuentes planas rectangulares.

En ambos desarrollos las cadenas de cálculo partieron de fuentes de *tracks* registradas durante una corrida del núcleo de RA10, en el código MCNP. Se utilizó un código desarrollado anteriormente para el sorteo de las partículas de la lista, sin procesamiento alguno, en simulaciones de McStas. A partir de allí inicia la utilización de fuentes de distribuciones, a través de detectores en dicho código.

1.3. McStas

McStas [4] es un paquete de simulación de instrumentos neutrónicos por método Monte Carlo, desarrollado en forma conjunta por DTU Physics y los institutos Laue Langevin (ILL), Paul Scherrer y Niels Bohr (NBI). Los modelos se construyen a través de un meta-lenguaje de alto nivel, en el cual se define una secuencia de componentes, incluyendo fuentes y detectores de partículas así como instrumentos neutrónicos. El programa convierte dicho *input* en un código en ISO-C con el cual se ejecuta la simulación y se obtienen los resultados. Se trata de un *software* de código abierto, es decir que tanto el motor de cálculo como la implementación de los componentes incluidos, en su mayoría desarrollada en lenguaje C, puede descargarse libremente de la página oficial de McStas.

Cada componente de McStas está definido en un archivo de texto, con una estructura que consta de bloques de código delimitados por comandos especiales, cuyos contenidos son en lenguaje C. Las transformaciones que sufre una partícula al atravesar cada componente están determinadas por los comandos allí definidos, con lo cual las posibilidades para instrumentos de óptica neutrónica son virtualmente ilimitadas. Sin embargo, esto hace que el código no sea útil para el cálculo de blindajes, ya que el modelado de materiales no está implementado.

Una de las grandes ventajas del carácter abierto de McStas es que pueden definirse nuevos componentes los cuales, en las simulaciones, resultan completamente equivalentes a los incluidos en el programa. Incluso es posible definir bibliotecas de desarrollo propio e incluirlas de forma simple en las definiciones de componentes. Estas características le dan al código una gran versatilidad, la cual resulta muy útil para el presente proyecto. En efecto, la implementación de fuentes de distribuciones en McStas, tanto para la detección como para la producción de partículas, se realizó mediante la definición de una biblioteca en lenguaje C y componentes como los descritos.

1.4. Tripoli

Tripoli [5] es una herramienta de transporte de radiación por el método Monte Carlo, desarrollada por el CEA (*Commissariat à l'énergie atomique et aux énergies alternatives*) en Francia. Su motor de cálculo resuelve la ecuación de Boltzmann tridimensional para neutrones, fotones, electrones y positrones, permitiendo la resolución tanto de problemas de criticidad como de fuente fija (cálculo de blindajes). Su biblioteca oficial de secciones eficaces y datos nucleares es CEAV5.1.1, basada en las evaluaciones europeas

JEFF-3.1.1. Entre sus características más destacables se incluyen: la implementación un método de reducción de varianza automático, la posibilidad de paralelización de los cálculos, y la posibilidad de incluir fuentes externas. Esta última funcionalidad se realiza mediante la definición en lenguaje C de una función que al ser llamada provea los parámetros que definen una partícula. La misma debe cumplir cierta estructura, y debe ser compilada con `gcc` con la bandera `-shared`, pudiendo incluirse código auxiliar o bibliotecas en el proceso, antes de la simulación.

La utilización de Tripoli se encuentra motivada por el hecho de que, desde 2017, Argentina pertenece a NEA (Nuclear Energy Agency), con lo cual tiene acceso a la licencia de los códigos incluidos en su repositorio, entre los que se encuentra dicha herramienta. El proyecto integrador de Ayala incluyó en buena medida la adquisición de conocimientos sobre su uso, y se pretende continuar el estudio sobre sus utilidades en las investigaciones locales.

En el presente proyecto se aprovechó la posibilidad de incorporar fuentes externas en Tripoli para la incorporación de los desarrollos de fuentes de distribuciones. Con respecto a la detección de partículas para su caracterización distribucional, se registraron listas de *tracks* en las zonas de interés, las cuales fueron procesadas posteriormente.

1.5. El Reactor RA10

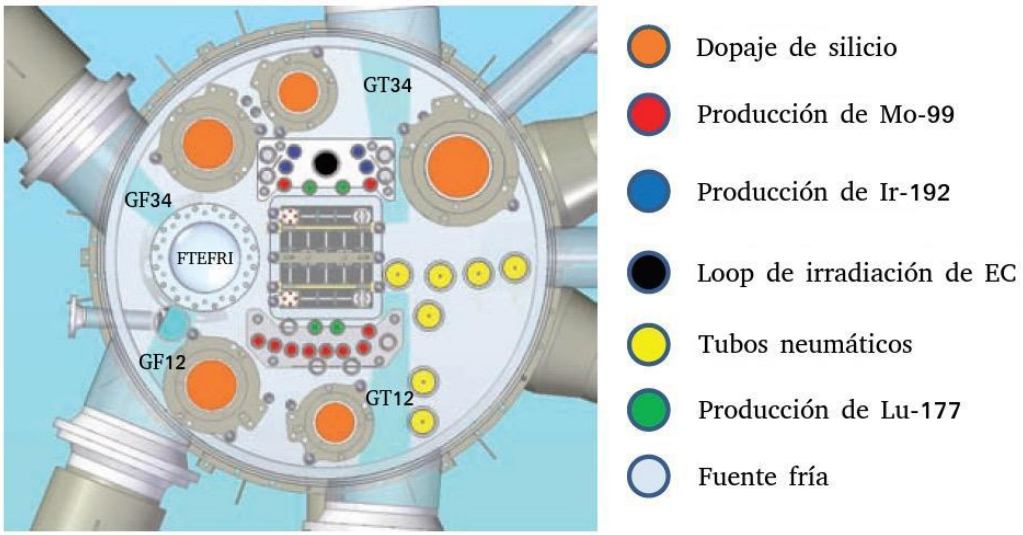


Figura 1.2: Vista horizontal de la pileta del RA10

El reactor RA10 es un proyecto de reactor nuclear multipropósito, el cual está siendo construido en el Centro Atómico Ezeiza, en Buenos Aires, Argentina. Se trata de un reactor de pileta tipo MTR, con potencia nominal de 30 MW. Contará con un núcleo con combustibles tipo placa refrigerado por agua liviana, rodeado por un tanque

de reflector de agua pesada, en el cual se encuentran la mayoría de las posiciones de irradiación, como se observa en la Figura 1.2.

Entre sus principales usos se encuentran la producción de radioisótopos, especialmente Mo-99, y la provisión de neutrones térmicos y fríos para diversos instrumentos de investigación. Se contará con una fuente fría de deuterio líquido, mientras que la fuente térmica será una región del tanque de reflector. Parte de los neutrones allí producidos serán transportados con guías neutrónicas por distancias de hasta 65 m, a través del *hall* de guías.

Capítulo 2

Desarrollo de herramientas computacionales

2.1. Arquitectura computacional

Como se explicó en el Capítulo 1, los motores de cálculo de las simulaciones Monte Carlo se comunican con archivos de formatos específicos, los cuales se emplean a modo de *input* y *output*. En los desarrollos anteriores de Fairhurst y Ayala los códigos empleados fueron McStas y Tripoli, y se utilizaron estos archivos para implementar el grabado y sorteo de fuentes de distribuciones, además de para inicialmente sortear fuentes de *tracks*. Los archivos principales en sus cadenas de cálculo, referidos a las fuentes de distribuciones, fueron:

- `EPA_DetectorX.comp`: Componente de McStas. Construye una fuente de distribuciones en base a una grilla provista, detecta y registra una dada cantidad de partículas, y guarda las distribuciones y libera la memoria alocada.
- `Source_builderX.comp`: Componente de McStas. Construye una fuente de distribuciones en base a un conjunto de distribuciones provisto, sortea una dada cantidad de partículas, y libera la memoria alocada.
- `source_n_p.c`: Componente de Tripoli. Construye una fuente de distribuciones en base a un conjunto de distribuciones provisto, sortea una dada cantidad de partículas, y libera la memoria alocada.

En los tres casos la geometría es plana rectangular, y todas comparten la estructura de discretización de variables, la cual fue descrita anteriormente. El primer objetivo de este proyecto es la implementación de un desarrollo similar sobre los espejos de una guía neutrónica. Esto significa construir detectores y fuentes de distribuciones que detecten, registren y sorteen los neutrones que escapan por cualquiera de las cuatro caras. Si bien

cada espejo tiene forma plana rectangular, se busca modelar a la guía en su conjunto, por lo cual la superficie de fuente tendrá forma de tubo de sección rectangular, y las partículas sorteadas nacerán con velocidad apuntando hacia el lado exterior.

2.2. Primeras implementaciones de detector y fuente sobre guía

Luego del aprendizaje e internalización del código McStas, se precedió al estudio de los códigos `EPA_DetectorX.comp`, `Source_builderX.comp` y `source_n_p.c` y su estructura de programación. Para un mejor entendimiento, durante esta etapa se reescribieron dichos códigos, uniformizando y añadiendo pequeños cambios a su metodología computacional.

A continuación se procedió a imitar dichos archivos adaptándolos a las caras de las guías, lo cual dio lugar a `Guide_leaks.comp` (similar a `EPA_DetectorX.comp`) y `Source_guide.c` (similar a `source_n_p.c`). Nótese que un componente análogo a `Source_builderX.comp`, a ser utilizado como fuente en McStas, no tiene sentido para una guía pues en este caso la fuente son los neutrones que escapan de ésta y son absorbidos en los blindajes, lo cual no es posible simular en dicho código.

El componente `Guide_leaks` se compone de las mismas tres etapas de su antecesor: creación de la fuente de distribuciones en base a grillas (inicialización), detección y registro de las partículas que escapan de la guía (simulación), y guardado de las distribuciones obtenidas y liberación de memoria (finalización), pero además durante la simulación propaga los neutrones a través de la guía. La secuencia de cálculo para cada partícula que ingresa a la guía es la siguiente: se calcula la posición del primer contacto con un espejo, se la propaga hasta dicha posición, se calcula la reflectividad, se guarda en las distribuciones la fracción no reflejada, se actualiza la velocidad y peso de la fracción reflejada, y se reinicia el ciclo con ésta última. Cuando se alcanza la salida de la guía, se propaga el neutrón hasta allí y la simulación continúa en el siguiente componente. El cálculo de la probabilidad de reflexión se basa en el m de la guía y otros parámetros de la curva de reflectividad, característica de los espejos, con ayuda de las funciones que incluye McStas para esta tarea.

Las variables elegidas para identificar a cada partícula se presentan en la Tabla 2.1 y se esquematizan en la Figura 2.2. La estructura de discretización sigue la misma lógica de macro y micro grupos de los códigos anteriores, aunque adaptada a la nueva geometría. En la Figura 2.1 se observa un diagrama de la estructura empleada.

El archivo `Source_guide.c`, el cual genera la fuente externa para Tripoli, sortea partículas en base a las distribuciones registradas por `Guide_leaks`. La secuencia de sorteo es análoga a la empleada en `Source_builderX`.

Var.	Definición (según espejo)				Uni.	Descripción
	$x+$	$y+$	$x-$	$y-$		
z	z				cm	Distancia en la dirección de propagación
$mirror$	$\frac{y+h/2}{h}$	$1+\frac{w/2-x}{w}$	$2+\frac{h/2-y}{h}$	$3+\frac{x+w/2}{w}$	1	Avance en la circulación transversal de la guía
E	$\frac{1}{2}m_n v^2$				MeV	Energía
μ	$\frac{v_x}{v}$	$\frac{v_y}{v}$	$\frac{-v_x}{v}$	$\frac{-v_y}{v}$	1	Coseno del ángulo de divergencia con respecto a la normal
ϕ	$tg^{-1}(\frac{-v_y}{v_z})$	$tg^{-1}(\frac{v_x}{v_z})$	$tg^{-1}(\frac{v_y}{v_z})$	$tg^{-1}(\frac{-v_x}{v_z})$	rad	Ángulo de deflección con respecto a la dirección de propagación

Tabla 2.1: Descripción de las variables empleadas para parametrizar las partículas escapando de una guía.

Cabe mencionar que los componentes construidos pueden utilizarse tanto con neutrones como con fotones. En la simulación en McStas, el código para sorteo de *tracks* `Ptrac_source_difra.comp` y el motor de cálculo no distinguen entre neutrones y fotones, por lo que simulan una u otra partícula "sin darse cuenta", aunque debe fijarse el valor $m = 0$ para todas las guías al simular fotones, para conservar el sentido físico. Al utilizarse `Source_guide.c` como fuente externa en Tripoli, debe indicarse en la implementación de la fuente si las partículas creadas serán neutrones o fotones, pues el formato de las distribuciones que ésta lee es el mismo.

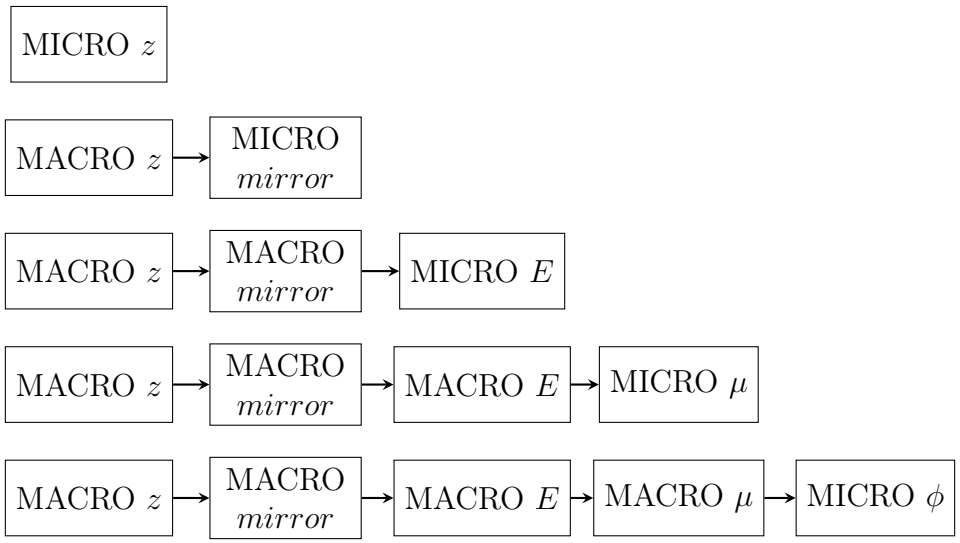


Figura 2.1: Esquema de la estructura de discretizaciones empleada para guías.

Es muy común que uno de los tramos en las guías de neutrones sea curvo, y en particular este es el caso de todas las guías en RA10. Por este motivo se procedió a implementar versiones generalizadas de los dos códigos anteriores, que consideraran una curvatura. Esto resultó en los códigos `Guide_leaks_curved.comp` y `Source_guide_curved.c`. En McStas el primer código propaga los neutrones de acuerdo a la geometría curva de la guía, es decir con paredes laterales cilíndricas, pero manteniendo en la simulación una virtual forma recta. Es decir que, en el *input* del programa, para las posiciones relativas entre la guía y demás componentes, la guía es recta, pero los neutrones en su interior se propagan como si ésta fuera curva, calculando correctamente las distribuciones de corriente a la salida, y de escapes a través de los espejos. En Tripoli, donde se simulan las estructuras que componen las guías y el blindaje, la geometría es genuinamente curva. La estructura de macro y micro grupos es la misma que para

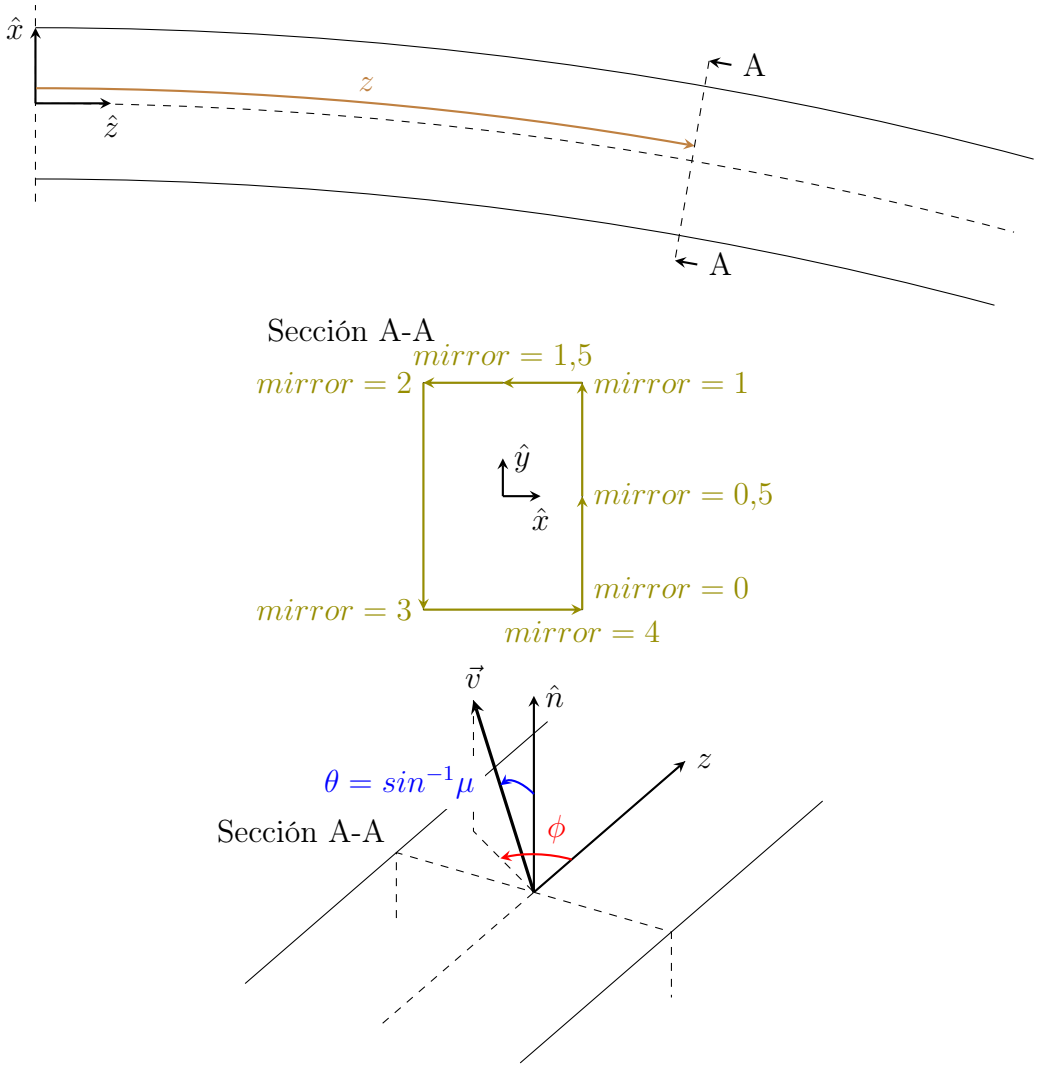


Figura 2.2: Esquema de las variables empleadas para identificar a las partículas escapando de guías curvas. En el caso de curvatura nula éstas coinciden con las correspondientes a guías rectas. La forma indicada para medir μ y ϕ es la misma para los 4 espejos, variando en cada caso el significado de $\hat{n}$.

guías rectas, aunque la variable z en este caso representa la distancia recorrida desde la entrada hasta una sección dada, a través del arco que forma el centro de la guía, como se muestra en la Figura 2.2.

2.3. Procesamiento de fuentes de *tracks* de Tripoli

Otro de los objetivos en el presente desarrollo es el de registrar fuentes de distribuciones en simulaciones de Tripoli, de modo similar a `EPA_DetectorX.comp`. En el código Tripoli no es posible crear nuevos componentes como en McStas, por lo que para dicha tarea se debe registrar una lista de *tracks* en la superficie de interés, y luego procesarla para calcular las distribuciones, las cuales serán leídas por `Source_builderX.comp` o `Source_guide.c` para generar nuevas partículas. La funcionalidad que se busca construir puede pensarse como una concatenación de los códigos `Ptrac_source_difra` y `EPA_DetectorX`, ya que se debe leer una lista de *tracks*, aunque proveniente de Tripoli y no de MCNP, identificar las partículas en ella, y una por una registrarlas en una fuente de distribuciones construida en base a un conjunto de grillas. Por último se deben guardar las distribuciones obtenidas. Cabe mencionar que el código a cumplir esta función no tiene ninguna restricción en su formato, pues no se comunica con ningún programa Monte Carlo.

Del objetivo arriba mencionado se implementó el código `EPA_Reader.py`, en lenguaje Python. Éste puede leer fuentes de *tracks* tanto en formato de MCNP como Tripoli, cargar todas las partículas en distribuciones, y guardar a éstas últimas con el mismo formato empleado en códigos anteriores para fuentes rectangulares. Es decir que los códigos `Source_builderX` y `source_n_p` pueden utilizar dichas distribuciones, sin posibilidad de distinguir si éstas fueron generadas por `EPA_Reader` o por `EPA_DetectorX`.

Para corroborar el correcto funcionamiento de los componentes desarrollados se modeló el haz GF1 de RA10. En ambos casos se empleó una fuente de *tracks* de neutrones proveniente de una simulación del núcleo en MCNP, registrada a la salida del conducto de extracción GF12, unos 50 cm antes de la entrada a las guías. Durante la corrida en McStas se compararon las corrientes en distintos puntos de la guía con un *benchmark* implementado con componentes incluidos en el programa, y se verificó que la corriente entrante sea igual a la suma de los escapes por los espejos y la corriente saliente. Además, se empleó `EPA_Reader.py` para generar distribuciones de la fuente. Los resultados obtenidos permitieron concluir que la implementación del los componentes fue correcta. En la simulación en Tripoli, por su parte, se implementó un modelo preliminar de las guías y los sistemas y blindajes alrededor de éstas, y se obtuvieron *tallies* de dosis ambiental por neutrones y por fotones *prompt*. También se registraron *tracks* en posiciones intermedias y con ellos se construyeron nuevas fuentes de distribuciones.

Esta implementación permitió probar la factibilidad de la cadena de cálculo. Si bien el modelado fue muy rudimentario, se pudo estudiar la practicidad, ventajas y desventajas del empleo de fuentes de distribuciones en el cálculo considerado. Además, permitió considerar las posibilidades de optimización de la estructura computacional, y evaluar la conveniencia de efectuar cambios en el esquema general de las fuentes de distribuciones implementadas.

2.4. Biblioteca de fuentes de distribuciones en lenguaje C

En todos los desarrollos propios o de Fairhurst o Ayala, hasta ahora utilizados, todas las directivas necesarias para implementar cada funcionalidad se encuentran en el archivo cuyo formato permite la comunicación con el código Monte Carlo. Éstos constan de entre 476 y 925 líneas, y el contenido de ellos depende de la geometría que el componente simula y el programa con el cual se comunica. Por este motivo, para construir nuevos tipos de fuentes, similares a los ya desarrollados, como se hizo con `Guide_leaks` y `Source_guide`, se debe reescribir esta extensión de código. Desde luego es útil iniciar el desarrollo desde una copia del componente que se desea modificar, pero los cambios a realizar están a lo largo de todo el archivo, con lo cual la tarea tomará un tiempo considerable.

Ante esta situación se decidió modificar la estructura hasta ahora empleada en la implementación de fuentes de distribución. En lugar de incluir todo el código necesario para la creación, uso y destrucción de las fuentes en cada código, se inició el desarrollo de una biblioteca que incluya las definiciones de objetos adecuados para la simulación de fuentes de distribuciones, y de funciones para su manejo, de modo de generalizar los conceptos presentes. Oportunamente, tanto McStas como Tripoli permiten la utilización de bibliotecas en C. En el caso de McStas es posible añadir librerías externas copiando los archivos `lib.h` (declaraciones de funciones) y `lib.c` (definiciones de funciones) a la carpeta `share` del programa, e incluir el comando `%include "lib"` en el bloque `SHARE` de la definición del componente, donde `lib` se debe reemplazar por el nombre de la librería a incluir. En el caso de Tripoli, las fuentes externas se escriben con el formato de una función en lenguaje C y se compilan antes de ejecutar la simulación, por lo que para incluir librerías simplemente se las compila junto con la definición de la fuente en un único archivo binario.

La librería creada se denominó `dsource`[1], compuesta por los archivos `dsource.c` y `dsource.h`. Los tres principales objetos, de tipo `struct` sobre los que se modeló la fuente de distribución fueron:

[1] En referencia a *distributional source*.

- `DSource`: Fuente de distribuciones. Modela la fuente de partículas en sí. Posee una lista con una distribución para cada variable y administra su uso.
- `Distrib`: Distribución. Contiene y administra el histograma de intensidad en función de una variable. Es una estructura de tipo árbol, donde cada distribución de tipo macro posee punteros a cada una de las distribuciones hijas.
- `Grid`: Grilla. Modela la estructura de discretización de una variable.

La librería `dsource` incluye las funciones requeridas para manipular estas tres estructuras, y efectuar sus tareas de manera general e independiente de la geometría que se esté modelando. Las funciones están agrupadas de acuerdo la estructura sobre la que operan, por lo cual todas poseen alguno de los prefijos `DS_`, `Distrib_` o `Grid_`, excepto algunas funciones auxiliares que se definieron para operaciones matemáticas específicas. Las funciones que permiten manipular y utilizar las fuentes de distribuciones, las cuales funcionan como interfaz de usuario, son las que operan sobre la estructura `DSource`, por lo que no se requiere en principio ninguna interacción entre el usuario y los objetos `Grid` y `Distrib`. Son las funciones que operan sobre `DSource` las que llaman a las funciones requeridas de `Distrib`, y éstas a su vez a las de `Grid`.

La idea es que la biblioteca modele el concepto de fuente de distribución de una manera general, permitiendo su aplicación a distintas geometrías y esquemas de grupos. De todos modos, para agilizar la utilización de cada geometría en particular (rectangular, circular, guía, etc.) se pueden crear extensiones a esta librería con funciones particulares para cada una de ellas. Cabe destacar que, con esta nueva implementación, las fuentes de distribuciones no tienen por qué ser superficiales, pudiendo ser volumétricas, lineales o puntuales.

A continuación se describen con más detalle la definición y utilidad de las tres estructuras de `dsource`. Para una documentación detallada véase el Apéndice A.

2.4.1. Conceptos generales

Para empezar, se debe esclarecer el modo de representar a las partículas. Para esto se distinguen dos formatos:

- Coordenadas absolutas: La partícula se determina por un vector de 9 componentes con el siguiente formato:

  ```
  [ipt, ekin, x, y, z, dx, dy, dz, p]
  ```

  Donde `ipt` es el índice de partícula, el cual vale 1 para neutrón y 2 para fotón, `ekin` es la energía cinética, en MeV, `[x, y, z]` son las coordenadas de posición, en cm, `[dx, dy, dz]` es el versor dirección, y `p` es el peso estadístico.

- Vector de parametrización: es un vector que mapea una geometría en particular. Por ejemplo, en el caso de una fuente plana rectangular como la implementada por Fairhurst y Ayala éste resulta:

  ```
  [ekin, x, y, mu, phi]
  ```

  La longitud de este vector depende de la geometría mapeada y de las restricciones sobre otros parámetros de las partículas de fuente. Cabe notar que el peso estadístico no figura en el vector de parametrización, sino que se maneja por separado.

Debido a que se busca simular fuentes en diferentes posiciones del espacio, e incluso con diferentes orientaciones, cada fuente posee un vector de traslación y uno de rotación. Esto introduce una nueva representación para las partículas: las coordenadas relativas. Éstas tienen el mismo formato de las coordenadas absolutas, pero la posición y dirección están medidas con respecto a los ejes solidarios a la fuente.

Para relacionar las coordenadas relativas con el vector de parametrización se debe contar con una función de parametrización, la cual, para una geometría dada, realiza la conversión en ambas direcciones. Desde luego, dicha función es completamente dependiente de la geometría a implementar, por lo que su definición no forma parte de la biblioteca `dsource`, sino que debe definirse en una librería adicional específica para cada geometría, cumpliendo ciertos requerimientos característicos de una función de parametrización.

El formato para representar la distribución multidimensional es con una lista (*array*) de árboles de distribuciones, de la misma longitud que el vector de parametrización. Cada nodo de dichos árboles es un objeto `Distrib`, y almacena las distribuciones sobre una variable de parametrización, siendo los nodos hoja las distribuciones de tipo micro. Por ejemplo, en la Figura 2.1, cada fila representa uno de estos árboles, y puede observarse que las variables correspondientes al final de cada árbol (hojas) son todas diferentes, y juntas conforman el vector de parametrización para la geometría de guía. También se observa que varios nodos macro se repiten (por ej.: $MACRO\ z$ aparece 4 veces). En estos casos crea una estructura `Distrib` para cada árbol, pero todas ellas emplean la misma grilla, definida por un objeto `Grid`. En todos los casos, la representación de una distribución sobre una variable incluye los histogramas de pesos (I), de pesos cuadrados (p^2) y de cuentas (N).

El tipo de partícula (neutrón o fotón) es único para cada fuente. De todos modos, como se explicará más adelante, es sencillo superponer varias fuentes de diferentes partículas en una misma simulación.

Las discretizaciones a utilizar para las diferentes variables de parametrización se especifican mediante archivos con formatos específicos. Una función se encarga de leer

dichos archivos y con ellos crear las grillas para cada variable. Luego crea distribuciones vacías con dichas grillas, las conecta construyendo los árboles y finalmente produce el objeto `DSource`, representando una fuente vacía. El proceso de creación de los árboles de distribuciones resulta muy dependiente del esquema de grupos empleado, por lo que la función encargada de construir fuentes vacías no está implementada en la biblioteca `dsource`, sino que se debe crear una para cada tipo de fuente implementado.

Las dos funciones principales para la utilización de fuentes de distribuciones son las de guardar y sortear, referidas respectivamente a la detección y producción de partículas. En el primero de los casos se transforman las coordenadas absolutas detectadas a vector de parametrización, se determina a qué grupos, en cada árbol de distribuciones, pertenece, y se suma el peso estadístico en la posición correspondiente de cada histograma. El proceso de sorteo consiste en obtener valores aleatorios de cada variable de modo que se respeten las distribuciones y correlaciones. Esto se logra llamando secuencialmente a cada árbol distribucional, cada uno para el sorteo de una variable del vector de parametrización.

Otras dos funciones de suma importancia para el uso de esta herramienta son las de registrar y cargar, es decir guardar en el disco duro las distribuciones obtenidas, y luego construir nuevamente la estructura de fuente con ellas. Los archivos empleados son documentos de texto, con un formato único e independiente de la geometría, uno por cada árbol de `Distrib` y uno para información general. El proceso de guardado de distribuciones es efectuado por funciones implementadas en `dsource`, aunque se deja libre la primera línea de cada archivo para información específica de cada geometría. La creación de fuentes resulta demasiado compleja para su implementación de forma única y general, al menos en los tiempos de este proyecto, debido a la gran variabilidad de estructuras que éstas pueden presentar. Sin embargo, se definieron ciertas funciones para asistir la lectura de distribuciones, pero la creación y conexión de las distribuciones que componen la fuente se debe efectuar de manera particular para cada geometría.

2.4.2. DSource

Este objeto representa la fuente en sí, y es el encargado de efectuar las tareas requeridas para su utilización. Incluye los siguientes parámetros, los cuales se esquematizan en la Figura 2.3:

- `name`: un *string* para el nombre de la fuente.
- `ipt`: índice del tipo de partícula que detecta/produce la fuente (1 para neutrón, 2 para fotón).
- `vlen`: longitud del vector de parametrización.

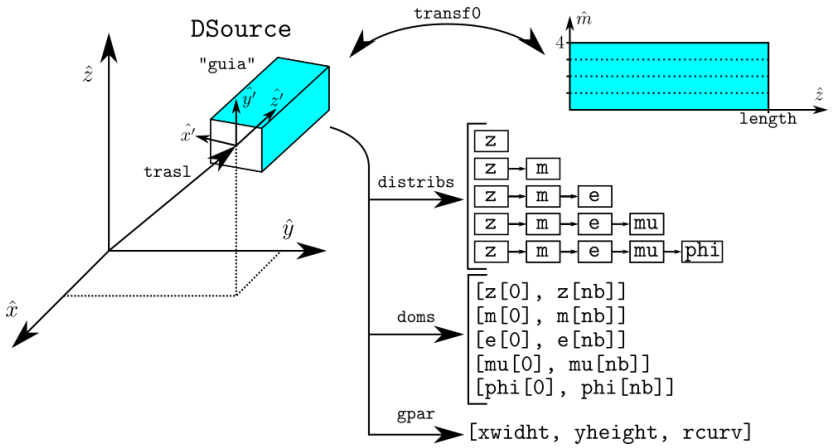


Figura 2.3: Esquema de los parámetros de la estructura `DSource`. Se representa, a modo de ejemplo, una fuente con geometría de guía, de nombre `guia`. Los parámetros `trasl` y `rot` (no representado) determinan la ubicación del sistema de coordenadas relativo a la guía $(\hat{x}', \hat{y}', \hat{z}')$, con respecto al absoluto $(\hat{x}, \hat{y}, \hat{z})$. La función `transf0` convierte entre coordenadas relativas y variables de parametrización, representadas por $\hat{z}$ y $\hat{m}$ (`mirror`). El parámetro `distribs` contiene las distribuciones en función de éstas últimas, `dom` contiene sus dominios, y `gpar` guarda parámetros geométricos de la guía (ancho, altura, curvatura).

- `distribs`: lista de los árboles de distribuciones que componen la distribución multidimensional.
- `doms`: lista de los intervalos de dominio sobre cada variable de parametrización.
- `transf0`: puntero a la función de parametrización.
- `trasl`: vector de traslación espacial de la fuente.
- `rot`: vector de rotación de la fuente, según el formato axial-angular.
- `gpar`: lista de parámetros adicionales necesarios para definir la geometría (ej.: altura, ancho).
- `gpar_len`: longitud de la lista de parámetros `gpar`.

A continuación se describen las funciones definidas para el manejo de la estructura `DSource`. Las principales se encuentran representadas en la Figura 2.4.

Las primeras funciones para el manejo de `DSource` son las de creación y destrucción, denominadas `DS_create` y `DS_destroy`. La primera de ellas recibe como argumentos cada uno de los parámetros que definen esta estructura, los cuales deben haber sido creados previamente, y con éstos aloca y crea una estructura `DSource`. Luego de utilizada una fuente, la función destrucción libera toda la memoria alocada. Como se explica en la Sección 2.5, la función `DS_destroy` puede incurrir en el error de doble liberación de memoria, por lo que es conveniente emplear funciones de destrucción específicas para cada geometría. Por otra parte, para una primera caracterización de una

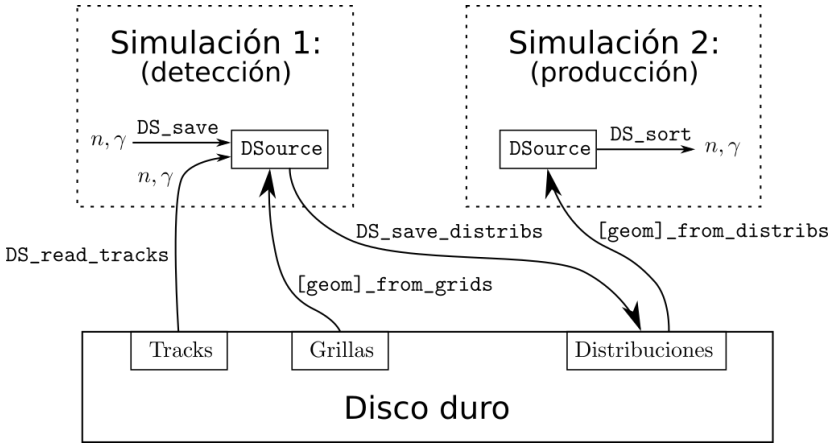


Figura 2.4: Esquema de las principales funciones de manejo de la estructura `DSource`, mediante las cuales se utilizan las fuentes de distribuciones. Inicialmente la fuente vacía se crea mediante `[geom]_from_grids`, donde `[geom]` es cada geometría en particular. Las distribuciones se obtienen cargando partículas a la fuente, lo cual puede lograrse mediante `DS_read_tracks`, o bien mediante `DS_save` durante una simulación. Una vez formadas las distribuciones la fuente se guarda mediante `DS_save_distribs`. Por último, en una nueva simulación la fuente se carga en base a las distribuciones guardadas, mediante `[geom]_from_distribs`, y se utiliza para sortear partículas con la función `DS_sort`.

fuente, las funciones `DS_I_tot`, `DS_p2_tot` y `DS_N_tot` devuelven la suma de pesos, pesos cuadrados y cuentas que ésta almacena.

La función `DS_transf` realiza la conversión entre coordenadas absolutas y vector de parametrización, en ambas direcciones. Esto incluye transformar las coordenadas de posición y dirección de los ejes absolutos a los relativos a la fuente, y luego llamar a la función de parametrización, o bien la secuencia inversa. El parámetro `gpar` es un argumento tanto de `DS_transf` como de `transf0`, ya que puede contener información necesaria para sus tareas.

El proceso de guardado es realizado por la función `DS_save`. Ésta recibe la asistencia de `DS_is_in`, la cual determina si la partícula a guardar pertenece al dominio de la fuente o no. En el caso afirmativo, se transforma a vector de parametrización y se llama a las funciones de guardado de cada árbol de `Distrib` en `distribs`, pasando como argumentos dicho vector y el peso de la partícula.

La producción de partículas es efectuado por `DS_sort`[2]. Para dicha tarea se crea un vector de parametrización, pero con el valor `INFINITY` (incluido en la biblioteca estándar `math`) en todos sus campos. Luego se llama, una por una, a las funciones de sorteo de cada árbol de distribuciones, con dicho vector como argumento. Cada árbol sortea el valor de la variable correspondiente al nodo hoja, o distribución micro, de éste, y utiliza los valores ya sorteados (aquellos distintos de `INFINITY`) para mantener la correlación, es decir determinar qué distribuciones micro usar. De este modo, luego de completada la lista `distribs`, el vector de parametrización tendrá valores reales en

[2]Entiéndase "sort" como abreviatura de "sortear", y no como "ordenar" en inglés.

todos sus campos, por lo que sólo se debe convertirlo a coordenadas absolutas para completar el sorteo.

La función `DS_save_distribs` realiza el guardado en el disco duro de una fuente ya grabada. Para ello se crea una carpeta y se crean en ella `vlen`+1 archivos, uno para información general, denominado `info.txt`, y los otros con cada distribución micro, denominados `distrib_d.txt`, donde `d` toma los valores de 1 a `vlen`. En el archivo de información se guardan las sumas totales de I, p^2 y N, el tipo de partícula, los vectores de traslación y rotación, y los parámetros geométricos de la fuente. En los archivos de distribuciones, como se mencionó anteriormente, se deja la primera línea para información específica de cada geometría. Las siguientes líneas, las cuales contienen todas las micro distribuciones del árbol, son escritas por la función análoga para `Distrib`. Los archivos mencionados se esquematizan en la Figura 2.5.

Como se explicó más arriba, no se cuenta con una función general para la creación de fuentes en base a las distribuciones guardadas. Sin embargo, se cuenta con una herramienta, `DS_read_info`, para la lectura del archivo de información, la cual fija los parámetros de una fuente ya creada.

Por último, se cuenta con una función para efectuar la lectura de fuentes de tracks, denominada `DS_read_tracks`. Ésta admite los formatos de MCNP (PTRAC) y Tripoli, y en ambos casos lee, una por una, todas las partículas de la lista (o la cantidad especificada) y las guarda en una fuente de distribuciones ya creada.

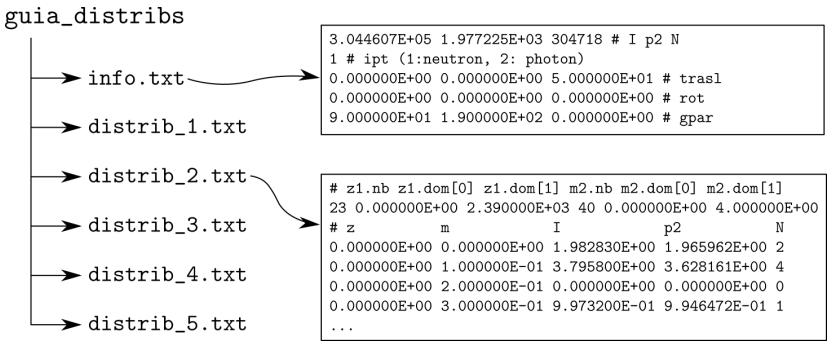


Figura 2.5: Esquema de los archivos utilizados para el guardado de fuentes de distribuciones en el disco duro. Se ejemplifica el caso de una fuente con geometría de guía. Se cuenta con un archivo `info.txt` con información general de la fuente, y una cantidad `vlen` (5 en este caso) de archivos con las distribuciones de cada árbol en `distribs`, todos dentro de una carpeta de nombre `[name]_distribs`. Se muestra el contenido de `info.txt` y `distrib_2.txt`, donde el texto luego de cada numeral (incluido el numeral) no está incluido realmente en los archivos, pero indica el significado de cada línea. El encabezamiento de los archivos de distribuciones contiene información de las grillas que componen el árbol, con formatos específicos para cada geometría. Para `distrib_2.txt` las variables son `z1` (z macro) y `m2` (*mirror* micro), y la información incluida es la cantidad de grupos y el dominio.

2.4.3. Distrib

Esta estructura simula la distribución de probabilidad sobre una única variable. Posee los siguientes parámetros, los cuales se esquematizan en la Figura 2.6:

- `name`: Nombre de la variable representada. Por ejemplo: `x`, `mu`, `ekin`, etc.
- `iv`: Índice de variable, es decir la posición que ocupa la variable representada en el vector de parametrización.
- `grid`: Estructura `Grid` con la grilla que maneja la discretización de la variable.
- `I`: Histograma de suma de pesos en función del grupo, con formato de *array*, de longitud igual a la cantidad de grupos.
- `p2`: Histograma de suma de pesos cuadrados en función del grupo.
- `N`: Histograma de cuentas en función del grupo.
- `cdf`: Función acumulativa de probabilidad. Se construye en base a `I`, por lo que también tiene formato de *array*, con la misma longitud. Se normaliza para que su último valor sea 1.
- `prev`: Puntero a la distribución madre.
- `nexts`: Lista de punteros a las distribuciones hijas, de longitud igual a la cantidad de grupos. Vacío para el caso de micro distribuciones.

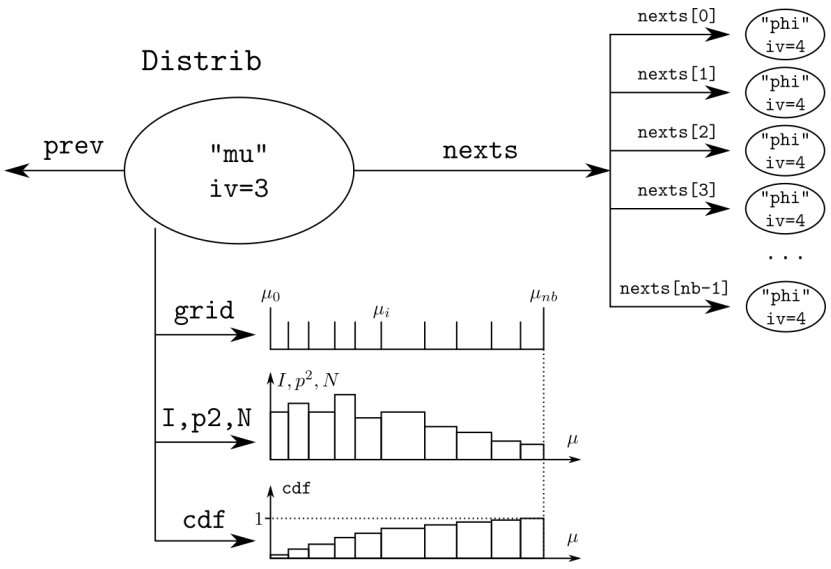


Figura 2.6: Esquema de los parámetros de la estructura `Distrib`. Se ejemplifica una distribución sobre la variable `"mu"`, con índice de variable 3. Al pertenecer a un árbol de distribuciones, posee un puntero `prev` a su `Distrib` madre, y como es una macro distribución también incluye una lista `nexts` de ramas con `Distrib` hijas. El parámetro `grid` contiene la discretización de μ entre μ_0 y μ_{nb}, con `nb` zonas o grupos. Los histogramas `I`, `p2` y `N` contienen las distribuciones, y `cdf` es la función acumulativa de `I`.

Análogamente al caso anterior, las primeras funciones requeridas son las de creación y destrucción, llamadas `Distrib_create` y `Distrib_destroy`. La primera de ellas crea una `Distrib` en base a sus parámetros previamente construidos, y la segunda libera toda la memoria asociada a la estructura. Se incluyen además dos herramientas para facilitar la construcción de árboles: si todas las `Distrib` de un mismo nivel (hijas, nietas, etc.) tienen la misma discretización, se puede crear el árbol directamente, dada una lista de los parámetros que definen cada nivel, mediante la función `Distrib_create_tree`; si, por el contrario, las diferentes hijas tienen diferentes grillas, la función `Distrib_create_nexts` conecta la `Distrib` raíz con la lista de ramas, las cuales deben ser creadas previamente, usualmente mediante un *loop*. Para obtener información descriptiva de una distribución, se incluyen las funciones `Distrib_I_tot`, `Distrib_p2_tot` y `Distrib_N_tot`, que devuelven las sumas de I, p^2 y N sobre todos los grupos, respectivamente, y otras dos que calculan la profundidad y la longitud total de una distribución, denominadas `Distrib_depth` y `Distrib_length`. La profundidad es la cantidad de niveles del árbol que componen las distribuciones, y la longitud total es la cantidad de últimos nodos (nodos hoja). Por otra parte, la función `Distrib_is_in` indica si un vector de parametrización pertenece a un árbol de distribuciones, o no.

Cuando se crea una distribución, no es necesario incluir inicialmente los histogramas. Si éstos no son provistos a la función de creación, el espacio para su almacenamiento no es alocado. Éste es el caso en la creación de fuentes vacías. Para iniciar su utilización, la función `Distrib_init` aloca la memoria para las distribuciones de I, p^2 y N y coloca un cero en todas las posiciones. Existe la posibilidad de que, luego de la inicialización, los nodos hijos no sólo ya estén inicializados sino que además ya contengan información, es decir que posean histogramas no nulos. En esta situación, es posible computar los histogramas de I, p^2 y N en base a las distribuciones hijas, mediante la función `Distrib_compute_I`. Para una distribución que ya posee histogramas no nulos, la función `Distrib_compute_cdf` computa la función acumulativa en base a la distribución I.

Las dos funciones principales para el uso de distribuciones son las de guardar y sortear, llamadas `Distrib_save` y `Distrib_sort`. Para la función de guardado, los argumentos son el peso estadístico y el vector de parametrización de la partícula a guardar, además de la `Distrib`. La función `DS_save` llama con dichos argumentos a cada distribución raíz de su lista de árboles, por lo que el objetivo de `Distrib_save` no es guardar la partícula en la `Distrib` en cuestión sino en todo el árbol del cual ésta es raíz. Por este motivo, únicamente las distribuciones micro guardan efectivamente las partículas en los histogramas de I, p^2 y N. Las distribuciones macro se limitan a definir a qué rama derivar el guardado, y sus distribuciones se mantienen sin siquiera ser inicializadas. El procedimiento es el siguiente: del vector de parametrización provisto se identifica la variable que corresponde a la `Distrib` utilizando el índice de variable `iv`,

y se determina a que grupo pertenece dicho valor. Si la lista de ramas `nexts` es no nula (`Distrib` macro), se llama a aquella en la posición indicada por el número de grupo de la partícula, con los mismos argumentos. Si, por el contrario, la distribución es micro, se procede a sumar el peso, peso cuadrado y la unidad en los histogramas correspondientes, controlando previamente que estén inicializados, llamando a `Distrib_init` si no fuera así.

De modo similar a lo que ocurre con la función de guardado, recién descrita, el comportamiento de `Distrib_sort` no es el mismo para distribuciones macro y micro. Como se mencionó anteriormente, la función `DS_sort` crea un vector con el valor `INFINITY` en todos sus campos y llama, en orden, a `Distrib_sort` sobre cada raíz de su lista, pasándoles dicho vector como argumento. Cada árbol de distribuciones debe sortear una de las variables del vector, lo cual efectúan las distribuciones micro. Nuevamente, las distribuciones macro se limitan a determinar a qué rama derivan la tarea. Sin embargo, de este último caso se distinguen dos posibilidades: o bien el valor en la posición `iv` del vector de parametrización es `INFINITY`, o bien es real. Si es real, es decir que ya ha sido sorteado, se determina a qué grupo pertenece, y se llama con los mismos argumentos a la función de sorteo sobre la rama correspondiente a dicho grupo. Si es `INFINITY`, se sortea su valor. Al tratarse de una distribución macro, es posible que se requiera computar sus distribuciones previamente. Tanto en este último caso como en el de una distribución micro, lo primero que se hace es controlar si existe la función acumulativa `cdf`, computándola en caso contrario. Luego se sortea el número de grupo empleando dicha función, lo cual consiste en sortear un número y distribuido uniformemente en $[0, 1)$ y determinar el número de grupo `g` para el cual `cdf[g-1]`$< y <$`cdf[g]`. Si la distribución es macro se llama recursivamente a la rama correspondiente, si ésta es micro se obtiene el valor de la variable sorteando un número uniforme dentro del grupo hallado previamente, y se lo guarda en el vector de parametrización, cumpliendo el objetivo requerido por `DS_sort`. Si se considera, por ejemplo, un esquema de grupos como el de la figura 2.1, el primer valor sorteado por una distribución micro será el de z, el cual será guardado en el vector de parametrización por la `Distrib` "micro z". Al ser llamada la raíz de cada uno de los siguientes árboles, se detectará que la variable z no es `INFINITY`, por lo que se determinará a qué macro grupo pertenece. Esta tarea debería ser efectuada por los 4 árboles cuya raíz es "macro z", a pesar de que poseen la misma grilla, por lo que el macro grupo será el mismo. Para evitar repetir dicho cálculo, existen otros dos argumentos de `Distrib_sort`, con los que `DS_sort` llama a cada árbol: una lista de punteros a `Grid`, y una lista de enteros, ambos de longitud `vlen`. La primera vez que una macro distribución detecta un valor real de su variable y calcula su macro grupo, durante el sorteo de una partícula, guarda en la posición `iv` de ambas listas el puntero a su grilla y el número de grupo hallado, respectivamente. De este modo, cuando la macro distribución correspondiente a la misma variable, en

otro árbol, detecta que existe un número de grupo ya calculado, evalúa si la grilla con la que se calculó es la misma que la `Distrib` posee. En caso afirmativo, no se repite el cálculo sino que se utiliza el valor ya calculado.

Para el guardado de las distribuciones ya obtenidas en el disco duro se cuenta con la función `Distrib_print`. La función `DS_save_distribs` crea el archivo donde se guardará cada distribución y escribe la primera línea, luego de lo cual llama a la raíz del árbol de distribuciones correspondiente, pasando como argumento la variable de tipo `FILE*` correspondiente a dicho archivo. El árbol de distribuciones se guarda en columnas, una para cada nivel del árbol, y tres para los histogramas micro de I, p^2 y N, como se observa en la Figura 2.5. Cada fila corresponde a un *bin* de dichos histogramas, y a su vez a un macro grupo de todas las variables macro. Por lo tanto cada línea posee los valores de inicio de cada grupo, primero los macro y luego el micro grupo (*bin*), y por último los valores de I, p^2 y N, todos éstos separados por espacios. Por lo tanto, cada línea debe ser escrita por todas las `Distrib` del árbol, de forma coordinada. Para ello se utilizan dos argumentos auxiliares, un entero `i` y una lista de tipo `double` llamada `line` de longitud igual a la profundidad del árbol menos uno. Cuando se llama `Distrib_print` sobre la raíz, `i` vale 0 y la lista ya está alocada. En general, cuando se llama una macro distribución se inicia un *loop* sobre todos sus grupos, y para cada uno de ellos guarda el valor de inicio del grupo en `line[i]`, y llama a la rama correspondiente con los argumentos `line` e `i+1`. De este modo, si la siguiente distribución también es macro efectuará un nuevo *loop* colocando el inicio de grupo en `line[i+1]`. Cuando se alcanza finalmente la micro distribución, `line` contendrá el inicio de un grupo de cada nivel macro. Se inicia nuevamente un *loop* sobre los grupos, en este caso micro grupos, y para cada uno de ellos se escriben en el archivo los números guardados en `line`, separados por espacios, seguidos por el valor de inicio del micro grupo, y los valores de I, p^2 y N. De este modo, luego de completados todos los *loops* recursivamente anidados, se habrá completado la escritura del archivo, con el formato descrito anteriormente.

De forma análoga a la función apenas descrita, `Distrib_read` lee una distribución guardada y guarda los valores de I, p^2, N, y de límites entre *bins* (grillas) en un árbol de distribuciones previamente creado. Al no existir una función general para la lectura de fuentes de distribuciones, la creación del objeto `DSource` al cual pertenecerá la lista de árboles, y el llamado a `Distrib_read` para cada uno de éstos, debe ser implementado para cada geometría o estructura de grupos en particular. La función recibe como argumentos la `Distrib` raíz del árbol y una variable de tipo `FILE*` con el archivo donde se guardó la distribución. La primera línea, donde se escribe información particular para cada geometría, ya debe haber sido leída. La lectura se realiza recursivamente, cada vez que `Distrib_read` es llamada sobre una `Distrib` que no es hoja, se realiza un *loop* llamando la misma función sobre todas las hijas. Inicialmente

todos los nodos del árbol tienen sus histogramas y sus grillas de discretización vacíos, sin alocar. Durante la lectura de la distribución se alocan y se guardan las grillas de todas las distribuciones (en el elemento `grid`), pero sólo se registran los histogramas de los nodos hoja, en concordancia con la metodología de `DS_save` al guardar partículas. Otros dos argumentos de `Distrib_read` son `line` y un entero `i`, similares a los de `Distrib_print`, excepto que en este caso `line` no es una lista de `double`s, sino de punteros a `double`, y en ella no se guardan los valores de inicio de cada grupo sino la dirección de memoria donde se guardarán éstos. Inicialmente la función es llamada con `i` valiendo 0 y `line` ya alocada, sobre la `Distrib` raíz. Si la lista de cortes entre grupos (perteneciente al objeto `Grid`) no está alocada, se aloca, y a continuación se controla si existen distribuciones hijas. En caso afirmativo se inicia un *loop* sobre todos los grupos, y en cada iteración se guarda en `line[i]` el puntero al valor de inicio del *bin* y se llama a `Distrib_read` sobre la rama correspondiente con `i+1`. Cuando se alcanza un nodo hoja, primero se controla si los histogramas de I, p^2 y N están alocados, alocándolos de no ser así, y luego se inicia un *loop* sobre los microgrupos. En cada uno de éstos se procede a la lectura de una línea completa del archivo. Las primeros `i` números se guardan en las direcciones de memoria presentes en `line`, el siguiente se convierte en el valor de inicio del microgrupo, y los últimos 3 se guardan en la posición correspondiente de `I`, `p2` y `N`. De este modo se logra la lectura completa del archivo, obteniéndose un árbol de distribuciones listo para su uso en una fuente. Por último, existe otro argumento de `Distrib_read`, de tipo entero, llamado `sort`. Si éste es 0, la lectura es la descrita hasta ahora, pero si es distinto de 0, cada vez que se guarda un valor de intensidad I se le suma a éste un número aleatorio, con distribución gaussiana y desviación estándar (σ) igual al error estadístico, obtenido como la raíz cuadrada de la suma de pesos cuadrados del *bin* correspondiente (histograma p^2). De este modo, si una fuente es cargada varias veces, las distribuciones obtenidas no serán idénticas sino que tendrán la dispersión característica de la fuente guardada.

2.4.4. Grid

La estructura de grilla modela la estructura de discretización de una única variable. Posee los siguientes parámetros:

- `bins`: Lista de los valores de corte entre grupos, de longitud igual a la cantidad de grupos más uno.
- `nb`: Cantidad de grupos.
- `dom`: Par de números con el límite inferior del primer grupo (`bins[0]`) y el límite superior del último grupo (`bins[nb]`).

- `form`: Palabra indicando el formato de la grilla, pudiendo ser `"lin"` (lineal), `"log"` (logarítmico) o `"tab"` (tabulado). En el caso lineal o logarítmico los valores de borde entre grupos, o sus logaritmos, respectivamente, serán equiespaciados, por lo que `bins` no es necesario, y vale `NULL`. En el caso tabulado, los límites entre grupos son los especificados por `bins`.

- `units`: Unidades de la variable discretizada, en formato `string`.

Nuevamente, las primeras dos funciones definidas para `Grid` son aquellas para creación y destrucción, llamadas respectivamente `Grid_create` y `Grid_destroy`. En la primera se aloca la memoria necesaria y se crea la estructura `Grid`, en base a los parámetros que la componen, mientras que en la otra se libera dicha memoria.

Las dos funciones centrales para utilizar esta estructura son las de evaluar e indexar, denominadas `Grid_eval` y `Grid_index`, respectivamente. La primera devuelve el valor de la variable en un límite entre grupos, dado un número entre cero y la cantidad de grupos. La segunda devuelve el número de grupo al que corresponde un valor provisto de la variable representada.

Para una mayor practicidad en el manejo de grillas, se determinó un formato para su escritura en un documento de texto. En base a un archivo con dicho formato, la función `Grid_read` lo interpreta y crea una estructura `Grid`. En este caso cada archivo contiene la información para crear una única grilla, la cual puede ser de tipo lineal o tabulada. Además, se creó también una función similar, llamada `Grid_read_list` que lee un conjunto (lista) de grillas de un único archivo, con un formato similar al anterior. Esta última función está especialmente pensada para casos como el empleado en las fuentes rectangulares por Fairhurst y Ayala, donde las discretizaciones de x e y son diferentes para cada macro grupo de energía, y puede resultar útil guardar todas éstas en un único archivo (uno para x y uno para y).

2.5. Geometrías implementadas

Como se explicó en la Sección 2.4, para completar el modelado de una fuente superficial con la herramienta `dsource` se deben definir algunas funciones particulares para cada geometría. Es decir que para implementar una geometría de fuente en particular, con una dada estructura de macro y micro grupos, se debe crear una biblioteca específica para ésta, la cual funciona como extensión de `dsource`. Las funciones que esta librería debe incluir son 5, y se describen a continuación, donde `[geom]` se debe reemplazar por el nombre de cada geometría. Algunas de ellas se esquematizan en la Figura 2.4.

- `[geom]_transf`: Transformación de parametrización. Tiene como argumentos un puntero a vector de coordenadas, un puntero a vector de parametrización, un

puntero al peso estadístico, un vector de parámetros geométricos y un entero denominado `inverse`. Este último puede valer 0 ó 1, e indica si se debe convertir de coordenadas a vector de parametrización y peso (`inverse`=0) o viceversa (`inverse`=1). En todos los casos se deben proveer los parámetros geométricos de la fuente (`gpar`), ya que pueden ser necesarios para la transformación. Nótese que también se debe conocer la ubicación espacial de la fuente. Dado que esta función será el parámetro `transf0` de `DSource`, ésta debería estar centrada en el origen, aunque esta condición no es mandatoria, y su significado puede variar para diferentes geometrías. Para fuentes superficiales o de dimensión inferior, no cualquier vector de coordenadas pertenecerá a la fuente (de hecho ésta forma una región de volumen nulo). En estos casos, en la definición de `[geom]_transf` se deberá establecer un criterio para decidir si un vector de coordenadas está lo suficientemente cerca de la superficie de fuente como para considerarse que pertenece a ésta. En caso contrario se deberá imprimir un mensaje de error o advertencia, y opcionalmente proyectar la partícula sobre la fuente y convertirla de todos modos.

- `[geom]_from_grids`: Creación de fuente vacía en base a grillas. Esta función debe crear todas las grillas empleadas por la fuente, leyéndolas de archivos con los formatos descritos en la Subsección 2.4.4, crear los árboles de distribuciones vacíos, y con ellos crear la estructura `DSource`.

- `[geom]_from_distribs`: Creación de fuente en base a distribuciones guardadas. Esta función lee el conjunto de archivos representados en la Figura 2.5, obteniendo de éstos los parámetros generales de la fuente, las grillas y las distribuciones. Para lograrlo es necesaria la información guardada en el encabezamiento de cada archivo `distrib_i.txt`, con un formato particular para cada geometría, adecuado a cada caso.

- `[geom]_headers`: Creación de encabezamientos para los archivos de distribuciones. Esta función es esencial para el guardado de distribuciones. Recibe como argumento una lista de `vlen` *strings* ya alocados, y escribe en cada uno de éstos el texto que se guardará en la primera línea de cada archivo `distrib_i.txt`. La información que debe contener es la cantidad de grupos y el dominio de cada grilla presente en el árbol de distribuciones correspondiente a cada archivo. La cantidad de grupos en los que se divide cada variable es elemental para poder leer posteriormente el archivo, y del dominio realmente sólo se necesita el valor superior, ya que en las demás líneas del archivo sólo se presenta el inicio de cada *bin*. Esta función debe ser específica para cada tipo de fuente ya que la cantidad de grillas diferentes para una misma variable en un mismo árbol puede variar.

- `[geom]_destroy`: Destrucción de la fuente. Esta función libera toda la memoria asociada a una fuente de distribuciones. Se requiere su implementación de manera particular para cada geometría ya que muchas veces varias `Distrib` usan una misma `Grid`, por lo que para no liberar dos veces una misma `Grid` se debe saber de antemano cuáles distribuciones comparten grilla.

Con esta estructura se implementaron 3 tipos de fuentes, denominados `window` (plana rectangular), `guide` (guía de neutrones) y `activ` (volumétrica), las cuales se describen a continuación. En todos los casos se emplearon grillas gruesas (macro) para las distribuciones de correlación (`nexts` no nulo) y grillas finas para las distribuciones finales (nodos hoja). Cabe destacar que la cantidad de código requerida para la implementación de cada nueva geometría, o nuevo esquema de macro y micro grupos, se ve considerablemente reducida con respecto a la metodología empleada anteriormente, en los códigos `Guide_leaks` y `Source_guide`.

2.5.1. Geometría "window"

La geometría `window` modela una fuente plana rectangular, como las empleadas por Fairhurst y Ayala en los códigos `EPA_DetectorX`, `Source_builderX` y `source_n_p`. Sus principales características se describen a continuación:

- Vector de parametrización: `[ekin, x, y, mu, phi]`.
  Con `ekin` la energía de la partícula en MeV, `x` e `y` su posición en cm, `mu` el coseno del ángulo entre su dirección de propagación $\hat{\Omega}$ y la dirección `z`, y `phi` el ángulo entre la dirección `x` y la proyección de $\hat{\Omega}$ sobre el plano `xy` en radianes, siguiendo la regla de la mano derecha.

- Estructura de macro y microgrupos: Se presenta en la Tabla 2.2.

- Grillas: Para cada una de las variables `ekin`, `mu` y `phi` se cuenta con una grilla gruesa (macro) y una fina (micro), cada una compartida por todas las `Distrib` del mismo tipo. Para `x` e `y` se cuenta con una grilla gruesa y una fina para cada macro grupo de energía.

- Parámetros geométricos: No posee.

macro `mu`	→	micro `ekin`							
macro `ekin`	→	micro `x`							
macro `ekin`	→	micro `x`	→	micro `y`					
macro `ekin`	→	macro `x`	→	macro `y`	→	micro `mu`			
macro `ekin`	→	macro `x`	→	macro `y`	→	macro `mu`	→	micro `phi`	

Tabla 2.2: Esquema de las grillas empleadas en las distribuciones de `window`.

2.5.2. Geometría "guide"

La geometría `guide` modela una fuente sobre las paredes de una guía de neutrones, como las empleadas en los códigos `Guide_leaks` y `Source_guide`, descritos en 2.2. Sus principales características se describen a continuación:

- Vector de parametrización: `[z, mirror, ekin, mu, phi]`.
  Con `z` la posición en la dirección de propagación de la guía en cm, `mirror` un índice entre 0 y 4 indicando la posición en la circulación transversal, `ekin` la energía de la partícula en MeV, `mu` el coseno del álgulo entre su dirección de propagación $\hat{\Omega}$ y la dirección normal al espejo, y `phi` el ángulo entre la dirección `z` y la proyección de $\hat{\Omega}$ sobre el plano del espejo en radianes, siguiendo la regla de la mano derecha (véase Figura 2.2).
- Estructura de macro y microgrupos: Se presenta en la Tabla 2.3.
- Grillas: Para cada una de las variables se cuenta con una grilla gruesa (macro) y una fina (micro), cada una compartida por todas las `Distrib` del mismo tipo.
- Parámetros geométricos: `[xwidth, yheight, rcurv]`, donde `xwidth` es el ancho en `x` de la guía, `yheight` la altura en `y`, y `rcurv` el radio de curvatura, todos en cm. La curvatura siempre es en la dirección de x negativa, para simular el caso contrario se debe rotar la guía 180° con respecto al eje `z`.

micro `z`								
macro `z`	→	micro `mirror`						
macro `z`	→	macro `mirror`	→	micro `ekin`				
macro `z`	→	macro `mirror`	→	macro `ekin`	→	micro `mu`		
macro `z`	→	macro `mirror`	→	macro `ekin`	→	macro `mu`	→	micro `phi`

Tabla 2.3: Esquema de las grillas empleadas en las distribuciones de `guide`.

2.5.3. Geometría "activ"

La geometría `activ` modela una fuente volumétrica, especialmente pensada para simular el decaimiento *gamma* de materiales activados neutrónicamente. En Tripoli dichos fotones no son producidos automáticamente, pero se pueden registrar *tallies* volumétricos de tasa de activación de un dado nucleido. Para esta geometría, además de las cinco funciones requeridas para toda geometría, se cuenta con un método denominado `Activ_from_tally`, el cual crea una fuente de distribuciones en base a un *tally* como los mencionados, y al espectro energético de emisión γ definido en un archivo CSV descargado de la base de datos de IAEA [6]. Sus principales características se describen a continuación:

- Vector de parametrización: `[ekin, z, x, y, mu, phi]`.
  Con `ekin` la energía de la partícula en MeV, `x`, `y` y `z` la posición de la partícula en cm, `mu` el coseno del ángulo entre su dirección de propagación $\hat{\Omega}$ y la dirección `z`, y `phi` el ángulo entre la dirección `x` y la proyección de $\hat{\Omega}$ sobre el plano `xy` en radianes, siguiendo la regla de la mano derecha.

- Estructura de macro y microgrupos: Se presenta en la Tabla 2.4.

- Grillas: Para cada una de las variables `x`, `y` y `z` se cuenta con una grilla uniforme, determinada al definir el *tally* en Tripoli, cada una compartida por todas las `Distrib` del mismo tipo. Para `ekin`, como las energías tienen valores discretos, se emplea una grilla con *bins* de tamaño cero (límite superior e inferior iguales) sobre cada valor de energía. Las grillas de `mu` y `phi` tienen un único *bin*, ya que la emisión es isotrópica.

- Parámetros geométricos: No posee.

ekin				
z				
z	$\rightarrow$	x		
z	$\rightarrow$	x	$\rightarrow$	y
mu				
phi				

Tabla 2.4: Esquema de las grillas empleadas en las distribuciones de `activ`

2.6. Integración de las herramientas desarrolladas con los códigos McStas y Tripoli

Para poder utilizar efectivamente la biblioteca desarrollada, es necesario crear archivos con los formatos adecuados para la comunicación con los códigos Monte Carlo a utilizar, es decir McStas y Tripoli, en adelante denominados archivos de comunicación. Considerando el rol de las fuentes de distribuciones esquematizado en la Figura 2.4, éstos deben funcionar como fuentes o detectores en McStas o Tripoli, teniendo en cuenta las limitaciones de dichos códigos para la programación de componentes. También existen los casos en los que no se requiere la comunicación con un motor Monte Carlo pero sí la conversión entre formatos, como por ejemplo de *tracks* a distribuciones. En la Figura 2.7 se presentan los formatos de fuentes de distribuciones y los motores de cálculo utilizados, con los archivos de comunicación representados como flechas entre los formatos que convierten.

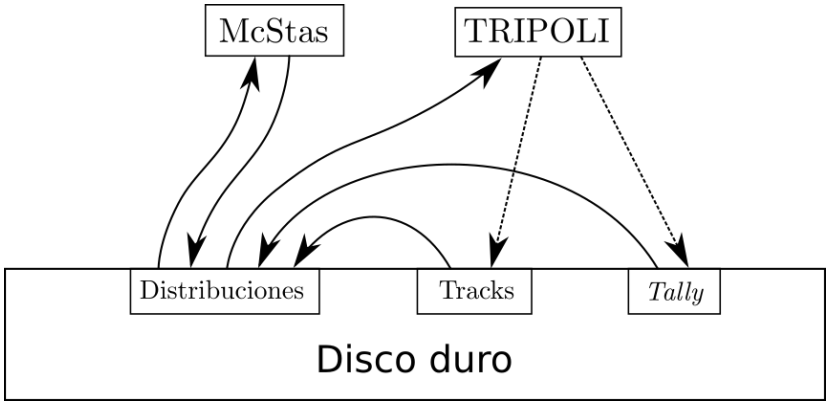


Figura 2.7: Esquema de la relación entre los códigos empleados entre sí y con las fuentes de distribuciones. Cada una de las flechas de línea continua representa un archivo de comunicación entre formatos, que ejecuta una serie de funciones de la biblioteca `dsource` con el formato requerido por los códigos o tipos de datos con los que interactúa. Las líneas punteadas representan la generación de archivos de *output* de corridas de Tripoli.

En el caso de McStas, al tratarse de un *software* de código abierto, es posible tanto modificar los componentes existentes como crear nuevos. Éstos se definen en archivos con sufijo `.comp`, escritos en un metalenguaje de McStas, pero con bloques de código en lenguaje C. También se puede incluir librerías de desarrollo propio en las simulaciones, y utilizar las funciones allí definidas en los mencionados bloques de lenguaje C. De este modo, en McStas tanto los detectores como las fuentes se implementan con el formato de componentes, lo cual le da gran libertad a la interacción entre la simulación y las acciones de lectura o escritura de archivos.

En Tripoli únicamente es posible programar en C las fuentes de partículas, no así los detectores. Para hacerlo se debe definir en dicho lenguaje una función que al ser llamada determine los valores de todas los parámetros que definen una partícula en Tripoli. La estructura computacional de dicha función resulta ser muy similar a la empleada en las fuentes de McStas, aunque en este caso no se utilice un metalenguaje. Para la detección de partículas existen dos posibilidades: el grabado de fuentes de *tracks* o la obtención de *tallies*. Ambos presentan sus resultados en archivos de *output* generados por Tripoli, por lo que el archivo de comunicación en este caso simplemente será un programa en C que los lea y convierta a distribuciones. Para agilizar los cálculos, se creó un archivo que condensa las conversiones de *tally* a distribuciones y de distribuciones a Tripoli, implementando directamente una fuente externa de Tripoli en base al *tally*.

En algunos casos, las tareas que debe efectuar el archivo de comunicación depende del tipo de fuente de distribuciones involucrado. En particular resulta muy diferente la implementación del detector de tipo `window` en McStas, que sólo registra las partículas, con aquel de tipo `guide`, que a la vez que detecta las fugas propaga los neutrones a lo largo de la guía. También existen casos en los que la comunicación sólo tiene sentido para un tipo de geometría. Este es el caso de la conversión de tallies volumétricos a

distribuciones, sólo empleados para fuentes de activación de tipo `activ`. Las fuentes en Tripoli resultan ser el caso más general, ya que pueden ser de cualquiera de los tres tipos de fuentes implementados, y pueden estar ubicadas en distintas posiciones y orientaciones del espacio. Por este motivo el archivo de comunicación de distribuciones a Tripoli desarrollado es algo más complejo que los demás, ya que permite la utilización simultánea de varias fuentes, que pueden tener diferentes geometrías y ubicaciones. Cada vez que se debe sortear una partícula, primero se determina aleatoriamente cuál fuente se utilizará, en base a las intensidades totales de éstas. En este proceso se incluyó un método de *source biasing*.

Para configurar los archivos de comunicación se dejan libres las primeras líneas a modo de *input*. En éstas se indican qué archivos se deberán leer, ya sea para obtener grillas, distribuciones, *tracks* o *tallies*. Dependiendo la tarea a realizar puede haber que indicar el tipo de partícula a detectar, o fijar la posición y rotación de la fuente, o bien elegir el tipo de fuente a modelar. Todas estas tareas se realizan definiendo variables en las primeras líneas de cada archivo, las cuales luego serán utilizadas durante la utilización de fuentes de distribuciones con el paquete `dsource`.

Para una fácil identificación de sus tareas, se empleó un formato único para los nombres de los archivos de comunicación. Cada nombre se compone por el nombre del código o archivo del cual parte, el carácter "2"[3], y el código o archivo al cual convierte, con el sufijo correspondiente (`.c` ó `.comp`). Si el archivo sólo tiene sentido para una geometría (`window`, `guide` o `activ`), antes del sufijo se añade `_window`, `_guide` o `_activ`, según corresponda. A continuación se listan todos los archivos de comunicación implementados:

- `Tracks2Distrib.c`
- `Distrib2McStas_window.comp`
- `McStas2Distrib_window.comp`
- `McStas2Distrib_guide.comp`
- `Distrib2Tripoli.c`
- `Tally2Distrib_activ.c`
- `Tally2Tripoli_activ.c`

Los archivos con sufijo `.comp` cumplen el formato de componente de McStas, mientras que aquellos con sufijo `.c` pueden o bien cumplir el formato de fuente externa de Tripoli (`Distrib2Tripoli.c`, `Tally2Tripoli_activ.c`), o bien constar de una función `main()` donde se ejecutan las funciones requeridas.

[3] En referencia a la similitud fonética entre *two* y *to* en inglés.

2.7. Biblioteca de fuentes de distribuciones en Python

En paralelo a la biblioteca implementada en lenguaje C, se efectuó un desarrollo equivalente en lenguaje Python. En ella se definieron todos los objetos y funciones descritas en la Sección 2.4, y las tres geometrías presentadas en la Sección 2.5, adaptadas a dicho lenguaje. Para las estructuras `DSource`, `Distrib` y `Grid` se utilizó el tipo de variable `class`, y las funciones de manejo fueron definidas como atributos de cada una de éstas. Como se explicó anteriormente, el lenguaje C es requerido para la comunicación con McStas y Tripoli, es decir que no es posible utilizar la biblioteca análoga en Python para la detección y producción de partículas con estos códigos. Las dos principales motivaciones para su desarrollo fueron los siguientes:

- Ensayar las estructuras y funciones para verificar su funcionamiento.
- Crear gráficos de las fuentes de distribuciones, con funciones integradas en la biblioteca.

Con respecto al primer punto, cabe mencionar que la implementación de `dsource` en Python se completó antes que su versión de C. Esto se debió a que, gracias a las mayores bondades que tiene Python con respecto a C, resultó útil para ensayar las estructuras y funciones que constituyen dicha librería. La corrección de errores de programación (*debugging*) es más rápida, por lo que resulta más ágil la determinación de los algoritmos más convenientes en cada caso, y la verificación de su funcionamiento. Una vez obtenida una versión funcional de las principales funciones de la biblioteca (`DS.save`, `DS.sort`, `DS.save_distribs`) se procedió a su traducción a C. Durante la etapa final de las tareas de desarrollo del presente proyecto, las últimas modificaciones al código se efectuaron directamente sobre la biblioteca útil para las fuentes de distribuciones, es decir aquella en C. Una vez cumplido el primer objetivo de la creación de la versión de Python, las modificaciones sobre ésta se centraron en el segundo de los objetivos.

Una de las ventajas de Python con respecto a C es la facilidad con la que se pueden crear gráficos, mediante la biblioteca `matplotlib.pyplot`. Esto fue aprovechado por una serie de funciones sobre la estructura `Distrib`, que permiten la creación rápida de gráficos de distintos tipos de las distribuciones `I`, `p2` y `N` de cada árbol de distribuciones en una fuente. Las funciones implementadas se describen a continuación. En todos los casos se trata de funciones definidas como atributos del objeto `Distrib`. Éstas crean los gráficos solicitados como figuras de `matplotlib`, y además devuelven la información en éstos presentada como listas de la librería `numpy`.

La principal herramienta para gráfico de distribuciones es `Distrib.plot`. Ésta crea una figura de `matplotlib` de la densidad de corriente en función de una única variable, la cual se calcula como el cociente entre los valores del histograma I y el ancho de cada *bin*. La función puede ser llamada sobre la distribución raíz de un árbol de

distribuciones, indicando cuál nodo graficar mediante el argumento `ind`. En éste se provee una lista de índices, que de estar vacía (valor por defecto) significa que la distribución a graficar es la del nodo actual. De no estarlo, se llamará recursivamente la misma función sobre la rama de índice igual al indicado por el primer número en la lista, pasando como argumento `ind` los números siguientes, de haberlos. Así, la lista `ind` inicial consiste en la secuencia de índices que identifican un nodo del árbol, cuya distribución se busca graficar. Si por ejemplo, en una guía con la estructura `guide` definida en la Subsección 2.5.2, se busca graficar la distribución de micro energía, para la macro zona 2 de `z`, y 3 de `mirror`, se deberá llamar la función `plot` sobre el árbol 3 de la lista de árboles `distribs`, con el argumento `ind=[2,3]`. Desde luego, también es posible llamar dicha función sobre el nodo deseado, accediendo a éste mediante los atributos `nexts` de las distribuciones macro, con `ind=[]`, obteniendo los mismos resultados. Nótese que es posible solicitar el gráfico de una distribución macro, en la cual los atributos `I`, `p2` y `N` no están inicializados. En ese caso se efectuará el cómputo de dichos histogramas con la función correspondiente para dicha tarea. Otro de los argumentos de `Distrib.plot` es `fact`, el cual se aplica como factor constante en el gráfico de distribuciones. Al llamarse la función sobre la distribución raíz, el valor por defecto es `fact=1`. Si el gráfico solicitado no es de dicho nodo (lista `ind` no vacía), al efectuar el llamado recursivo de `plot` sobre la rama correspondiente, se divide el valor de `fact` por el ancho del *bin* antes de pasarlo, de modo que en el gráfico final la distribución esté dividida por todos los intervalos empleados para el registro del histograma. Desde luego, inicialmente se puede proveer un valor distinto de la unidad, lo cual puede ser útil para convertir las corrientes a unidades físicas. La función de graficado posee una serie de argumentos adicionales para configurar otras opciones del gráfico, entre las que se encuentran: barras de error (obtenidas del histograma `p2`), normalización, escalas de los ejes, etiquetas, etc.

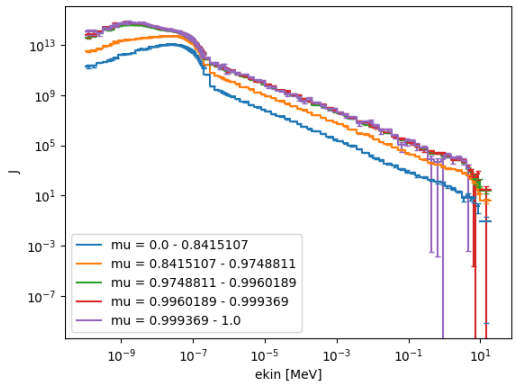
Otras dos funciones incluidas en la librería para graficar distribuciones de un objeto `Distrib` son `Distrib.plot_I` y `Distrib.plot_cdf`. Su funcionamiento es similar al de `plot`, sólo que las variables a graficar en este caso son `I` y `cdf`. En el primero de los casos esto significa que los valores del histograma `I` se presentan sin dividirse por el ancho del *bin*. Por el otro lado, `plot_cdf` grafica los valores de la función acumulativa de probabilidad, computándola previamente si es requerido. En ambas funciones está presente el argumento `ind`, con el mismo significado descrito anteriormente.

La última función para la creación de gráficos es `Distrib.plot_2D`. Ésta debe ser llamada sobre una distribución con lista de ramas no nula, ya que crea gráficos bidimensionales de corriente en función de dos variables (la de la distribución madre y la de las hijas). La cantidad graficada es la misma que la de `plot`, sólo que en este caso sus valores son representados mediante un color. La función empleada para la impresión en pantalla de las figuras creadas es `matplotlib.pyplot.imshow`, a la cual

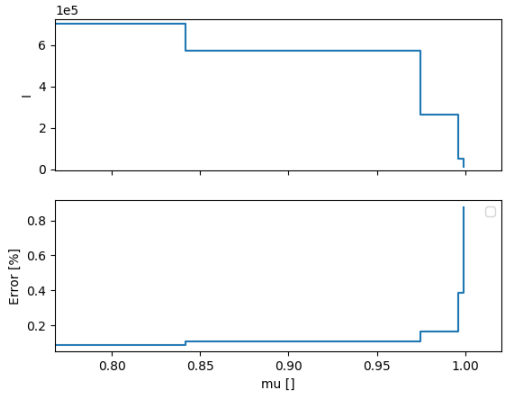
se le provee la distribución bidimensional a graficar en forma de matriz.

Las herramientas de graficación desarrolladas en la biblioteca `dsource` en Python son de gran utilidad para la caracterización de fuentes de distribuciones, tanto de neutrones como de fotones, ya que se emplea la misma estructura computacional al graficarlas que al utilizarlas en los cálculos con McStas y Tripoli. Pero existe una posibilidad adicional que brinda esta herramienta, asociada al principal objetivo de este proyecto, es decir el cálculo de tasas de dosis ambiental. Dicha cantidad física se calcula durante las corridas en Tripoli mediante *tallies* volumétricos, con un formato de *output* idéntico al empleado al calcular tasas de activación en materiales. Dado que en la librería de Python se implementó la lectura de fuentes volumétricas de activación directamente de los archivos de *output* de Tripoli, método análogo a la función `Activ_from_tally` descrita en la Subsección 2.5.3, es posible utilizar estas funciones para leer mapas tridimensionales de tasa de dosis, creando fácilmente gráficos en una y dos dimensiones, facilitando la observación de su distribución espacial.

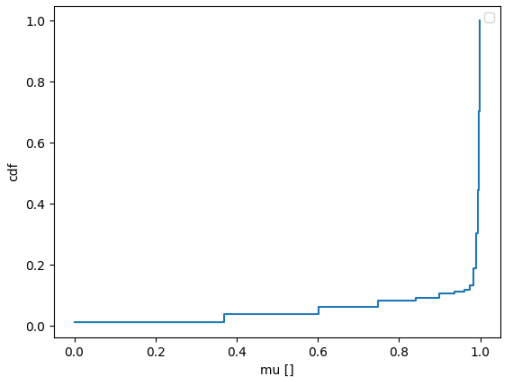
En la Figura 2.8 se muestran ejemplos de gráficos creados con las funciones descritas.



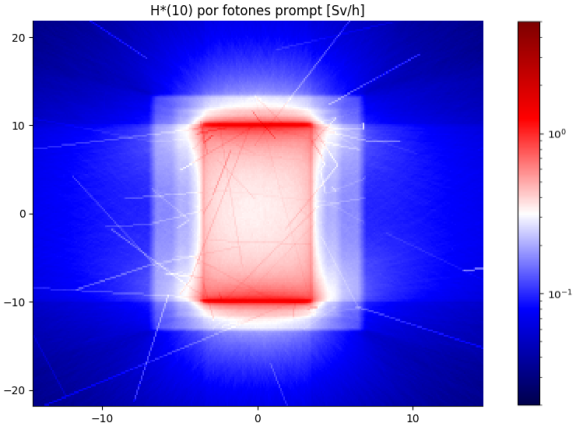
(a) Espectros energéticos en unidades arbitrarias, para distintos macro grupos de `mu`



(b) Intensidades de los macro grupos de `mu`, y los errores estadísticos relativos.



(c) Función acumulativa de intensidad de `mu` micro.



(d) Dosis por fotones *prompt* en una sección transversal de una guía de neutrones.

Figura 2.8: Ejemplos de figuras generadas con la biblioteca `dsource` implementada en lenguaje Python.

Capítulo 3

Cálculo de blindajes en un modelo conceptual de guía de neutrones

Con el objetivo de validar la herramienta desarrollada, y a la vez demostrar su aplicabilidad a cálculos de blindajes, se modeló en McStas y Tripoli el haz GF1 de RA10. Dicha guía de neutrones fue simulada en [2] con McStas, por lo que la parte de la simulación a efectuar con este código resulta ser un *benchmark*, lo cual permitirá corroborar la validez de los resultados obtenidos utilizando la biblioteca `dsource`. En particular, el principal resultado a comparar es la corriente a la salida de la guía. En el exterior de la guía, donde se efectúa el cálculo de blindajes, se construyó en Tripoli un modelo conceptual de un búnker y *hall* de guías, únicamente conteniendo a GF1.

El objetivo propuesto para el cálculo de blindajes es demostrar que la dosis en todo el *hall* de guías es inferior a un valor umbral adecuado para la protección radiológica de los trabajadores, fijado en 3 µSv/h según la legislación argentina [1]. Desde luego, si esto no se cumple con el diseño original se deberán realizar modificaciones, por ejemplo en el espesor de las paredes de hormigón del búnker de guías, o de los recubrimientos de parafina borada, hasta obtener un diseño adecuado, y preferentemente optimizado. Se supone que dentro del búnker de guías no ingresarán personas con el reactor en operación, por lo que no hay restricción para la dosis en su interior, ni tampoco dentro del blindaje exterior de la guía o el *beam catcher*.

3.1. Consideraciones generales sobre los cálculos con fuentes de distribuciones

Las fuentes de *tracks* que dan inicio a la simulación fueron grabadas en una corrida del núcleo y tanque de reflector de RA10, con MCNP6, de tipo criticidad. La cantidad de neutrones generados en el núcleo en dicha simulación fue de $NPS = 10^9$, mientras que en la superficie donde se grabaron los *tracks* se registraron intensidades (suma

de pesos estadísticos) de $I^{n}_{fuente} = 1,60 \cdot 10^{6}$ neutrones y $I^{\gamma}_{fuente} = 8,73 \cdot 10^{5}$ fotones. Según [2] la cantidad de neutrones nacidos por fisión en el núcleo del RA10 será de $FP = 2,3 \cdot 10^{18}\, n/s$, con lo cual la corriente en la superficie de las fuentes de *tracks* se puede calcular como:

$$J_{fuente} = FP \cdot \frac{I_{fuente}}{NPS} \tag{3.1}$$

Donde J_{fuente} e I_{fuente} pueden referirse tanto a neutrones como a fotones. La misma fórmula es válida si J_{fuente} e I_{fuente} representan las corrientes parciales en ciertos intervalos de energía, posición y ángulo (*bins*), es decir que si se multiplican las distribuciones registradas por el factor FP/NPS se obtienen las corrientes en unidades físicas. Si éstas se grafican con las herramientas de la biblioteca `dsource` en Python, se debe utilizar dicho factor como argumento `fact`.

Cada vez que se utilizan fuentes de distribuciones, la cantidad de partículas nacidas en una superficie es mayor que las que se detectaron en la simulación anterior, lo cual permite mejorar la estadística en las zonas lejanas al núcleo. Por lo tanto, cada vez que se utiliza esta técnica se deberá añadir un factor de la forma:

$$f_{DSource} = \frac{I^{det}_{DSource}}{I^{prod}_{DSource}} \tag{3.2}$$

Donde $I^{det}_{DSource}$ es la intensidad total detectada en el grabado de la fuente de distribuciones y $I^{prod}_{DSource}$ es la cantidad de partículas generadas con dicha fuente.

En una simulación compuesta por N corridas, desde el núcleo de RA10 hasta un detector de tipo fuente de distribuciones (`DSource`), la fórmula general empleada para la conversión de una intensidad registrada en corriente resulta:

$$J = FP \cdot \left(\prod_{i=1}^{N-1} \frac{I^{det}_{i}}{I^{prod}_{i-1}} \right) \cdot \frac{I}{I^{prod}_{N-1}} = \frac{FP}{NPS} \cdot \left(\prod_{i=1}^{N-1} \frac{I^{det}_{i}}{I^{prod}_{i}} \right) \cdot I \tag{3.3}$$

Donde I y J son la intensidad (suma de pesos estadísticos) y la corriente (en partículas/seg), totales o parciales, en la última `DSource` de detección, I^{det}_{i} e I^{prod}_{i} son las intensidades detectadas y producidas por la i-ésima fuente de distribuciones empleada ($I^{prod}_{0} = NPS$).

En el caso de la activación neutrónica, ésta es registrada por Tripoli con unidades de reacciones en cada celda por partícula de fuente, y convertida luego a fuente de distribuciones de tipo `activ`. La cantidad registrada resulta ser el cociente $I^{det}_{i}/I^{prod}_{i-1}$, siendo I^{det}_{i} la intensidad de activaciones e I^{prod}_{i-1} la intensidad neutrónica de fuente de la i-ésima corrida. En el equilibrio (estado estacionario), la tasa de activaciones equivale a la tasa de emisiones. En conclusión, incluso empleando tanto fuentes de distribución superficiales como volumétricas, la fórmula 3.3 sigue siendo válida.

Por último, en la última corrida de la simulación se realiza un cálculo de dosis

ambiental H*(10), la cual es reportada por Tripoli en µSv por partícula de fuente. Por lo tanto, para obtener la dosis en µSv/h se debe multiplicar este valor por la corriente en partículas por segundo en la fuente de esta región de simulación:

$$H^*(10)\left[\frac{\mu Sv}{h}\right] = 3600\frac{s}{h}\cdot\frac{FP}{NPS}\cdot\left(\prod_{i=1}^{N-1}\frac{I_i^{det}}{I_i^{prod}}\right)\cdot H^*(10)\left[\frac{\mu Sv}{part.gen.}\right] \tag{3.4}$$

El modelo a implementar está compuesto por varios cálculos de dosis, todos calculados con la fórmula 3.4, para obtener por separado las distintas fuentes de dosis en el *hall* de guías. Éstas son:

- Neutrones de fuente
- Fotones de fuente
- Fotones de reacciones (n,γ) *prompt*
- Fotones de activación

La restricción impuesta en la dosis ambiental en el *hall* de guías opera sobre la suma de las dosis parciales debidas a estas fuentes:

$$H^*(10) = H^*(10)_n + H^*(10)_{\gamma f} + H^*(10)_{\gamma p} + H^*(10)_{\gamma a} < H^*(10)_{max} = 3\ \frac{\mu Sv}{h} \tag{3.5}$$

El cálculo de blindajes apunta a la simulación de la operación nominal del reactor, en estado estacionario. En particular, las dosis por activación calculadas corresponden al estado de equilibrio entre generación de nucleidos radiactivos de interés (Al-28, Fe-59) y decaimientos. Es decir que todos los resultados presentados en las siguiente secciones corresponden a la dicha condición del reactor. Sin embargo, es posible utilizar los resultados estacionarios para obtener información sobre transitorios. Las tres primeras fuentes de dosis listadas anteriormente pueden considerarse proporcionales a la potencia del reactor instante a instante, ya que sus efectos dinámicos ocurren en escalas de tiempo muy inferiores al segundo. La activación, sin embargo, sí tiene escalas de tiempo apreciables, del orden de minutos o superiores. En las simulaciones se obtiene la distribución de tasa volumétrica de activaciones A la cual, al depender directamente del flujo neutrónico, puede considerarse instante a instante proporcional a la potencia. Por lo tanto, la evolución de la dosis por activación puede obtenerse de la ecuación de conservación de cada nucleido radiactivo relevante:

$$\frac{dN_j}{dt}(t) = A_j(t) - \lambda_j N_j(t) \tag{3.6}$$

$$\Rightarrow N_j(t) = N_j(t=0)e^{-\lambda_j} + \int_0^t A_j(t')e^{-\lambda_j(t-t')}dt' \tag{3.7}$$

Donde λ_j es la constante de decaimiento del nucleido j. Por lo tanto, puede obtenerse la dosis ambiental a tiempo t, dada una evolución de potencia $P(t) = P_0 f(t)$, en función de las dosis obtenidas para la potencia nominal P_0 en estado estacionario, denotadas $H^*(10)_n$, $H^*(10)_{\gamma f}$, $H^*(10)_{\gamma p}$ y $H^*(10)^{(j)}_{\gamma a}$, siendo j cada unos de los emisores de fotones de activación considerados, mediante la siguiente fórmula:

$$H^*(10)(t) = (H^*(10)_n + H^*(10)_{\gamma f} + H^*(10)_{\gamma p}) \cdot f(t) + \sum_j H^*(10)^{(j)}_{\gamma a} \left(e^{-\lambda_j} + \lambda_j \int_0^t f(t') e^{-\lambda_j (t-t')} dt' \right) \quad (3.8)$$

A modo de ejemplo, en el caso de un apagado brusco del reactor, aproximando una potencia tipo escalón descendente: $P(t) = P_0 \ \forall t < 0$, $P(t) = 0 \ \forall t > 0$, se obtiene la siguiente evolución de dosis ambiental:

$$H^*(10)(t) = \sum_j H^*(10)^{(j)}_{\gamma a} e^{-\lambda_j}, \ \forall t > 0 \quad (3.9)$$

Cabe mencionar que en la presente simulación no se considera la activación dentro del núcleo y tanque de reflector, ya que en dicha zona sólo se realiza una corrida en MCNP, incluyendo sólo neutrones y fotones *prompt*. Esto justifica que los fotones de fuente sean en todo momento proporcionales a la potencia, pero representa una aproximación considerable.

3.2. Descripción de los modelos implementados en McStas y Tripoli

En el código McStas el modelo se limita al interior de la guía. Inicia en la superficie donde se grabaron las fuentes de *tracks* de las que parte la simulación, ubicada antes de la entrada a la guía. Las partículas que no entran a ésta son descartadas de la simulación, mientras la que sí lo hacen son propagadas por su interior, perdiendo una fracción de su peso estadístico en cada reflexión en un espejo, variable según su energía y ángulo. Las guías están caracterizadas por su valor de m y otros parámetros de la curva de reflectividad. La simulación en McStas se restringe a su interior, pero al estar implementadas tanto las guías como la superficie de salida con fuentes de distribuciones las corrientes de salida quedan registradas en los archivos de distribuciones correspondientes, que funcionarán como *input* para Tripoli. Como se explicó en la Sección 1.3, McStas está diseñado para la propagación de neutrones, pero es posible utilizarlo para propagar fotones sin que haya diferencia para el programa, recordando que se deben modificar los parámetros de las guías (fijar m=0) para simular correctamente la inter-

acción con dichas partículas. De este modo las guías simplemente absorben todos los fotones que tocan sus paredes, pero al ser a su vez fuentes de distribuciones registran las corrientes salientes a ser introducidas en el modelo de Tripoli.

En la Figura 3.1 se muestra un esquema de los tres tramos de los que se compone la guía GF1. En la Tabla 3.1 se muestran las dimensiones de los mismos, así como el valor del m de los espejos. En toda la longitud de la guía la sección transversal es de 7 cm de ancho por 20 cm de alto.

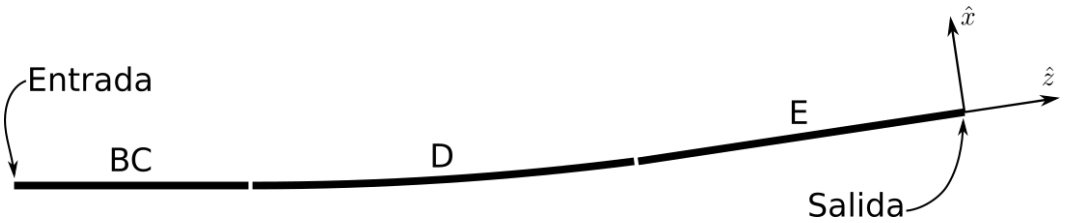


Figura 3.1: Esquema de los tres tramos de los que se compone la guía GF1.

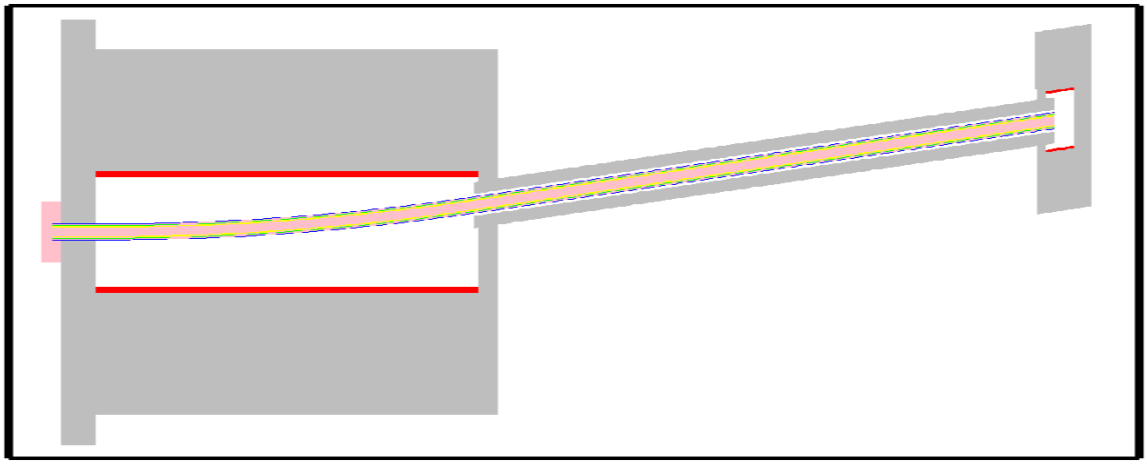
Tramo	Longitud [m]	m espejos				Radio de curvatura [m]
		$x+$	$x-$	$y+$	$y-$	
BC	3	3	3	3	3	-
D	20,2	3	3,5	3	3	929
E	32	3,5	3,5	3	3	-

Tabla 3.1: Dimensiones y propiedades de los espejos en los 3 tramos del haz GF1.

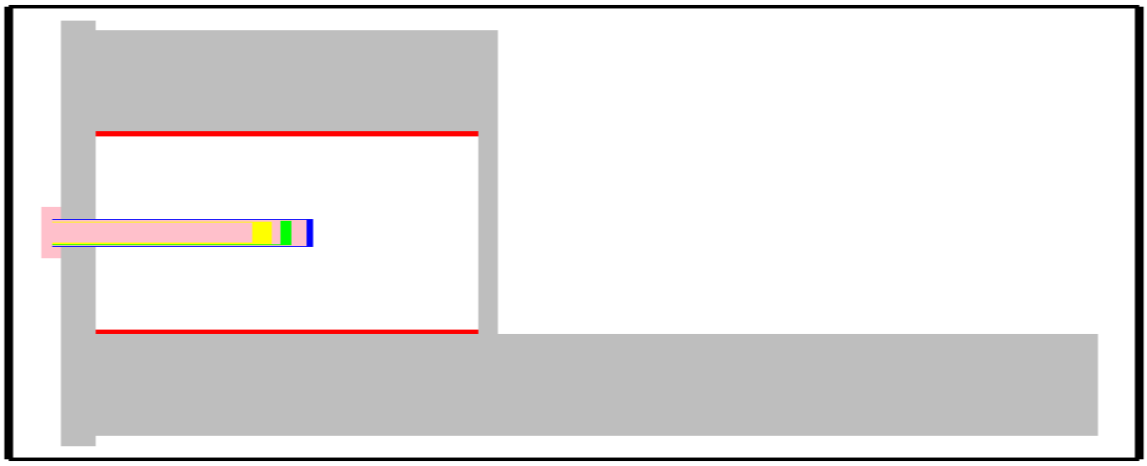
En la Figura 3.2 se presentan algunas imágenes de cortes del modelo implementado en Tripoli, el cual se basa en el diseño del *hall* de guías del RA10, y otras tecnologías típicas de guías de neutrones. Los materiales utilizados se definen en la Tabla 3.2. Las aleaciones de acero y de aluminio, así como el vidrio borado, representan materiales comerciales genéricos de cada tipo, mientras que el hormigón pesado simula un concreto experimental de alta densidad [7]. Cabe mencionar que el modelo realizado, junto con sus materiales, no busca retratar exactamente el *hall* de guías del RA10, sino que consiste en un modelo conceptual con el principal objetivo de demostrar la utilidad de la herramienta realizada.

Desde el punto de vista de los componentes simulados, el modelo construido en Tripoli consta de los siguientes elementos:

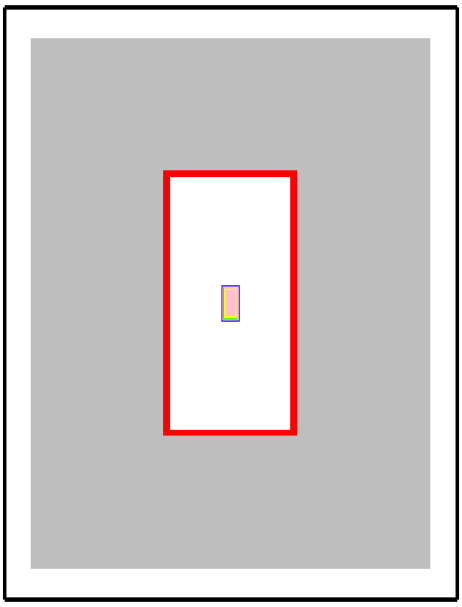
- Guía de neutrones. Ésta se compone por una región interior de sección rectangular rellena con helio a presión subatmosférica, donde se propagan los neutrones, y una serie de capas también de sección rectangular, cada una con espesor constante, las cuales se describen a continuación.
    - Espejos de vidrio borado con un recubrimiento de níquel y titanio en su parte interior.



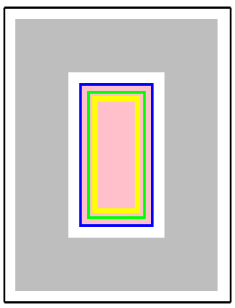
(a) Corte en el plano y=0 (horizontal).



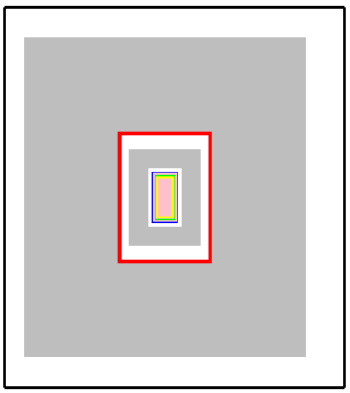
(b) Corte en el plano x=0.



(c) Corte transversal del búnker.



(d) Corte transversal del blindaje exterior.



(e) Corte transversal del *beam catcher*.

Figura 3.2: Distintos cortes del modelo conceptual de búnker y *hall* de guías, implementado en Tripoli.

Material	Recubr. de espejos	Vidrio borado	Hormigón pesado	Aleación de aluminio	Acero	Parafina borada
Densidad (g/cm^3)	8,106	2,86	5,87	2,7	8,05	0,9
Elemento/ nucleido	Fracción másica (%)					
H-nat			1,6712			14
B-nat		3,34				5
C-nat					0,15	81
N-nat					0,1	
O-nat		45,11	36,69			
Na-nat		7,61				
Mg-nat				0,9		
Al-nat			0,328889	95,8		
Si-nat		32,42	1,021	1	0,75	
P-nat					0,045	
S-nat					0,03	
K-nat		6,38				
Ca-nat			4,682			
Ti-nat	45,1			0,55		
Cr-nat			15,921	0,25	17	
Mn-nat				0,7	2	
Fe-nat			20,915	0,5	72,925	
Ni-58	54,9					
Ni-nat					7	
Cu-nat				0,1		
Zn-nat				0,2		
W-nat			18,548			

Tabla 3.2: Composición isotópica de los materiales usados en el modelo implementado en Tripoli. Los elementos con el sufijo "-nat" representan la suma de todos los isótopos del mismo, con sus abundancias naturales según la información del programa de secciones eficaces GALILEE.

- Huelgo de helio.
- Capa de aluminio, en representación del sistema de alineación de las guías.
- Huelgo de helio.
- Sistema de vacío de las guías, compuesto por un tubo de acero inoxidable.

En la Figura 3.3 se muestra una sección de la guía, generada en Tripoli, con las dimensiones de cada parte de la misma.

- *Block* del reactor. Consiste en una pared de hormigón pesado, atravesada por el tubo del sistema de vació de las guías. Del lado del reactor sólo se modeló un volumen de helio rodeando a las estructuras de la guía, representado el tubo de vuelo del haz.
- Búnker de guías. Es un recinto rodeado por paredes de hormigón pesado, con un recubrimiento de parafina borada en su interior. Son 5 paredes: dos laterales, una superior, una inferior y una frontal, las cuales junto con el *block* del reactor encierran un recinto relleno de aire en su composición natural, dentro del cual se encuentra la última parte del primer tramo recto de la guía GF1 y todo el tramo curvo. La pared frontal se ubica inmediatamente después del inicio del segundo tramo recto, siendo atravesada por éste. Las dimensiones del búnker se presentan en la Figura 3.4.
- Blindaje exterior de la guía. En el exterior del búnker, la guía está rodeada por una nueva capa, en este caso de hormigón pesado. Dicho blindaje va desde el inicio del segundo tramo recto de la guía hasta su fin, atravesando la pared frontal del búnker de guías.

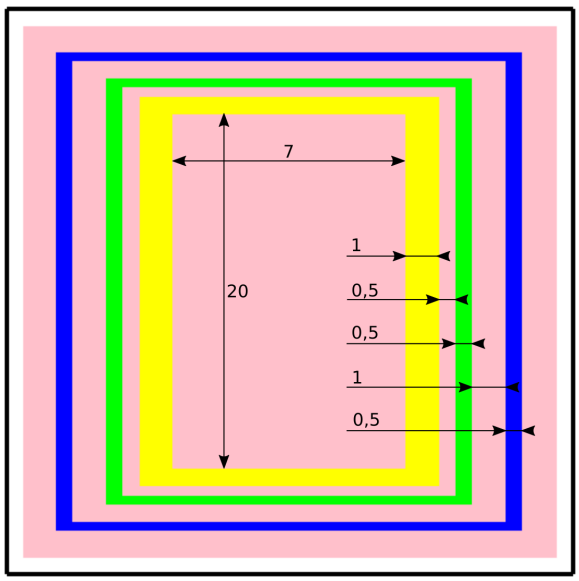


Figura 3.3: Dimensiones del modelo de guías y sistemas asociados, en cm.

- *Beam catcher*. Es una estructura de hormigón pesado, con un recubrimiento interior de parafina borada, que encierra el último tramo de la guía, blindando la radiación causada por la corriente de neutrones térmicos salientes. Sus dimensiones se observan en la Figura 3.5. Desde luego, a la salida de las guías en realidad se ubicarán los instrumentos de experimentación, por lo que el *beam catcher* debe considerarse una simple solución genérica para el blindaje de la salida de las guías.

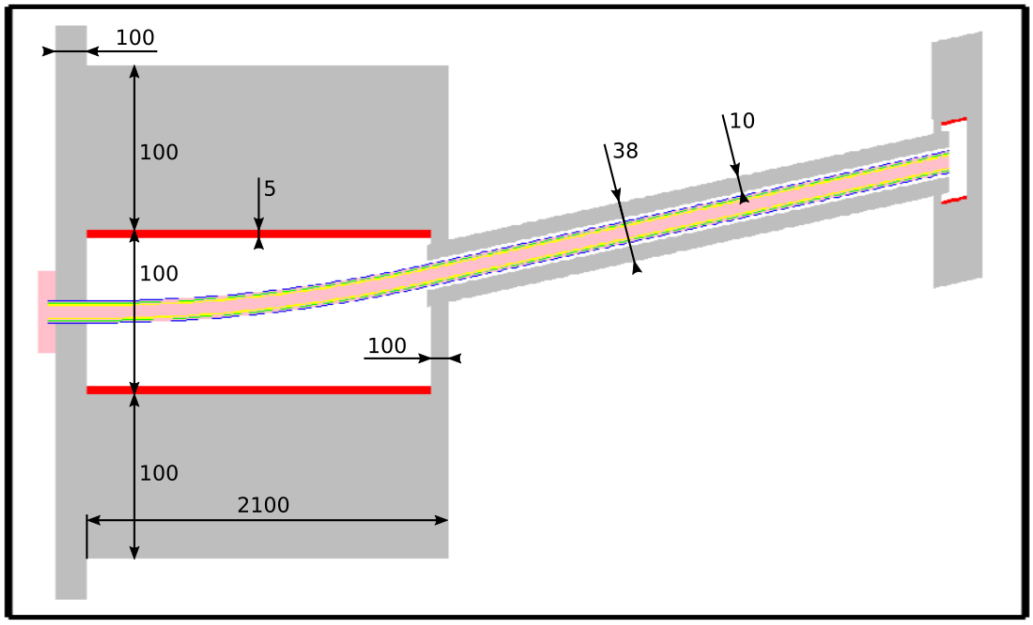
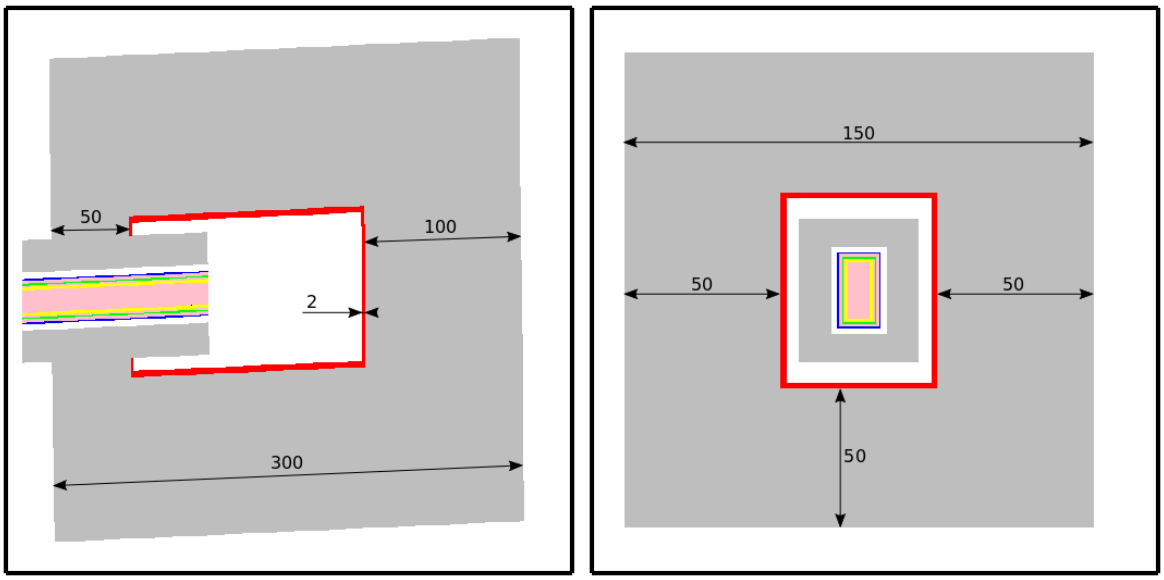


Figura 3.4: Dimensiones de los principales elementos presentes en el *hall* de guías, en cm. En la dirección vertical los espesores de cada blindaje son los mismos, y la altura del interior del búnker es de 200 cm.



(a) Corte horizontal. **(b)** Corte transversal.

Figura 3.5: Dimensiones del modelo de *beam catcher*, en cm.

- *Hall* de guías, es decir el exterior del búnker. Consta de un piso de hormigón pesado que continúa la pared inferior del búnker, aunque sin recubrimiento. Sobre éste se encuentran el búnker de guías, el blindaje exterior y el *beam catcher*, conteniendo a la guía, y sumergidos en un volumen de aire cuyos límites determinan el fin de la simulación. En la Figura 3.4 se presentan las dimensiones de los principales componentes en el *hall*.

3.3. Esquema de la simulación

Desde el punto de vista de la utilización de fuentes de distribuciones, la simulación consta de las siguientes etapas:

- **Simulación del tubo de vuelo en McStas**. Los neutrones y fotones nacen en una superficie de sección circular que corta el tubo de vuelo que alimenta los haces GF1 y GF2, y son propagados hasta la entrada del primero de éstos. Las partículas que llegan allí son registradas por una fuente de distribuciones de tipo `window`, mientras que las que no lo hacen son descartadas. Por las razones explicadas posteriormente en la Subsección 3.5.2, se eligió no utilizar una fuente de distribuciones para la producción de partículas en esta etapa, sino que directamente se propagó los neutrones y fotones de la lista de *tracks*.

- **Simulación del interior de la guía en McStas**. Los neutrones y fotones son producidos con la fuente de distribuciones a la entrada de la guía registrada en la etapa anterior, y son propagados hasta escapar completamente o alcanzar el final. Se utilizan tres fuentes de tipo `guide` para los tres tramos de la guía, y una de tipo `window` a la salida, garantizando que todas las partículas producidas serán registradas al salir del volumen de simulación.

- **Simulación del interior del búnker de guías en Tripoli**. Los neutrones y fotones nacen de las fuentes de distribuciones registradas en las caras de las guías en la etapa anterior. La región de interés en este caso es el interior del búnker, por lo cual se simulan los neutrones que escaparon de la guía en sus dos primeros tramos (interior del búnker), pero también aquellos registrados en el último de éstos, ya que por *scattering* podrían volver a entrar a dicha región. No se incluye la fuente registrada a la salida de la guía ya que se considera despreciable su aporte al flujo dentro del búnker. En el caso de los fotones todos los escapes se dan dentro de éste, ya que la curvatura de la guía impide a dichas partículas llegar al segundo tramo recto. La simulación abarca todo el *hall*, por lo que con tiempo de cálculo suficiente podrían obtenerse valores de dosis afuera del búnker, pero para reducir dichos tiempos se procede a realizar una nueva etapa de conversión a fuentes de

distribuciones, en este caso en la una superficie cuyos lados se encuentran a 5 cm de profundidad en el búnker, y cuyo frente es la interfaz entre el recubrimiento de parafina y la pared de hormigón. Para ello se registran en listas de *tracks* todas las partículas que alcanzan dicha interfaz, y luego se convierten a dos fuentes de distribuciones: una de tipo `window`, sobre la pared frontal, y otra de tipo `guide` sobre las paredes laterales, superior e inferior. En esta etapa también se registran los mapas de dosis dentro del búnker por neutrones, fotones *prompt* y fotones de fuente, y se calculan los *tallies* de activación de Al-28 y Fe-59 (consideradas las activaciones más relevantes) en los componentes de hierro y aluminio de las guías. Con los mapas de activación obtenidos se crean fuentes de tipo `activ` con los espectros de emisión correspondientes, y se realiza una nueva corrida, donde obtiene el mapa de dosis por fotones de activación y se registran las corrientes a través de las paredes del búnker, del mismo modo que con los neutrones, fotones *prompt* y fotones de fuente.

- **Simulación del *hall* de guías en Tripoli**. En este caso, a la geometría modelada descrita anteriormente (Figura 3.2) se le sustrajo el volumen correspondiente al interior del búnker, como se observa en la Figura 3.6, imponiendo en el límite con dicha región la condición de contorno de fuga, es decir que las partículas que ingresen a ésta son descartadas. Esto es necesario para conservar las condiciones de contorno correctas entre las simulaciones. Las simulaciones en esta región se dividieron en dos partes:

  - **Radiación proveniente del búnker**: Se utilizan como fuentes las distribuciones de corriente saliendo del interior del búnker, registradas en la etapa anterior. Las fuentes de neutrones, fotones *prompt*, fotones de fuente y fotones de activación se corren por separado, excepto los neutrones y fotones *prompt*, que sí se simulan juntos, obteniendo en cada caso los mapas de dosis. En esta etapa se consideran despreciables las nuevas activaciones fuera del búnker.

  - **Radiación proveniente del tramo E de la guía**: La fuente inicial es la corriente de neutrones escapando del tramo E de la guía. Esta lleva al cálculo de dosis por neutrones y por fotones *prompt*, así como a un *tally* de activación en el aluminio y hierro de los componentes de la guía. Con la fuente de activación registrada se efectúa una nueva simulación. En este caso no se emplean fotones de fuente ya que, como se mencionó anteriormente, ninguno alcanza este tramo de la guía. No se sortean partículas desde la salida de la guía pues, como se mencionó anteriormente, el *beam catcher* modelado no es realmente representativo de los instrumentos en esta posición, quedando

el cálculo de blindajes del componente correspondiente fuera del alcance del proyecto.

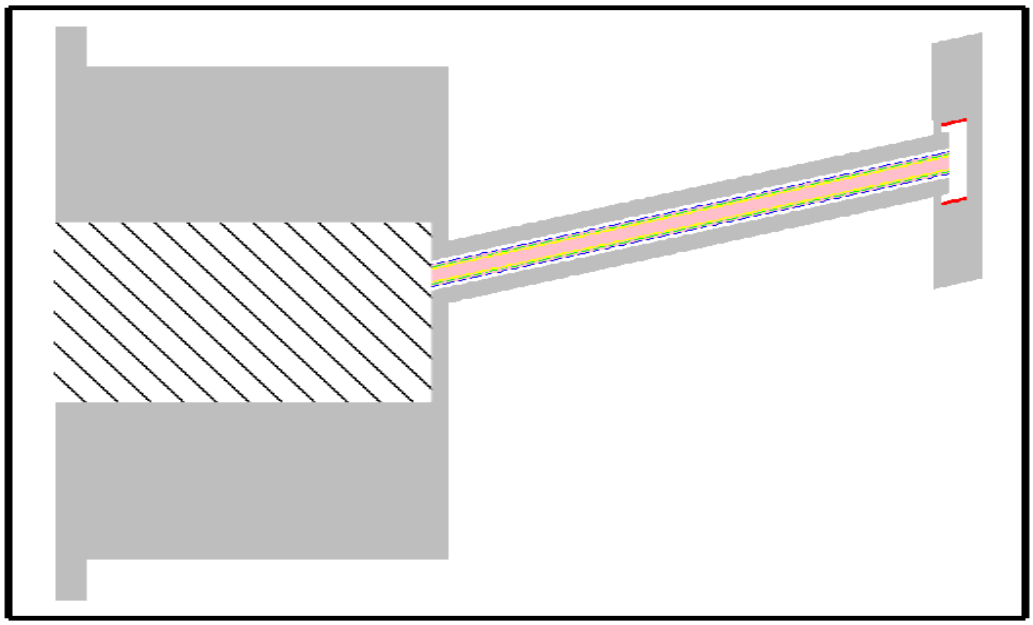


Figura 3.6: Modelo empleado para la simulación en Tripoli del exterior del búnker (*hall* de guías). La zona rayada indica el volumen sustraído de la simulación.

3.4. Caracterización de las fuentes de *tracks* utilizadas

Como se mencionó anteriormente, las simulaciones realizadas parten de dos fuentes de *tracks* registradas a la entrada de los haces GF1 y GF2 en una corrida de MCNP6, una de neutrones y otra de fotones. En la Figura 3.7 se observa la ubicación de la superficie donde se grabaron dichas fuentes, dentro del tanque de reflector del RA10. El formato de las listas de *tracks* empleado fue PTRAC, con lo cual es posible su lectura mediante la biblioteca `dsource`, desarrollada en el presente proyecto, más precisamente a través de la función `DS_read_tracks`. Si bien, como se mencionó en la sección anterior, en la primera etapa de simulación las partículas fueron sorteadas sin ser convertidas a fuentes de distribuciones, de todos modos se realizó dicha conversión para la caracterización de las fuentes de *tracks*. Se empleó una fuente de tipo `window`, de 52 cm de ancho y 52 cm de alto, aunque las partículas grabadas se ubican dentro de un círculo contenido exactamente en dicha región cuadrada.

La cantidad de partículas en las listas de *tracks* fue de $1,60{\cdot}10^{6}$ neutrones y $8,73{\cdot}10^{5}$ fotones. Considerando la fórmula de conversión a corriente en 3.1, y el área de la fuente,

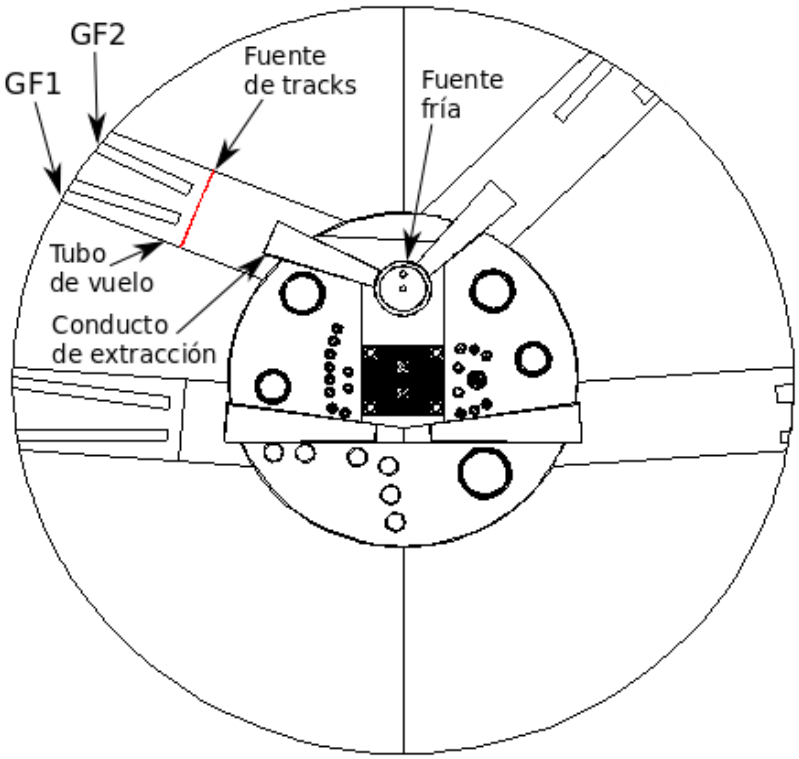


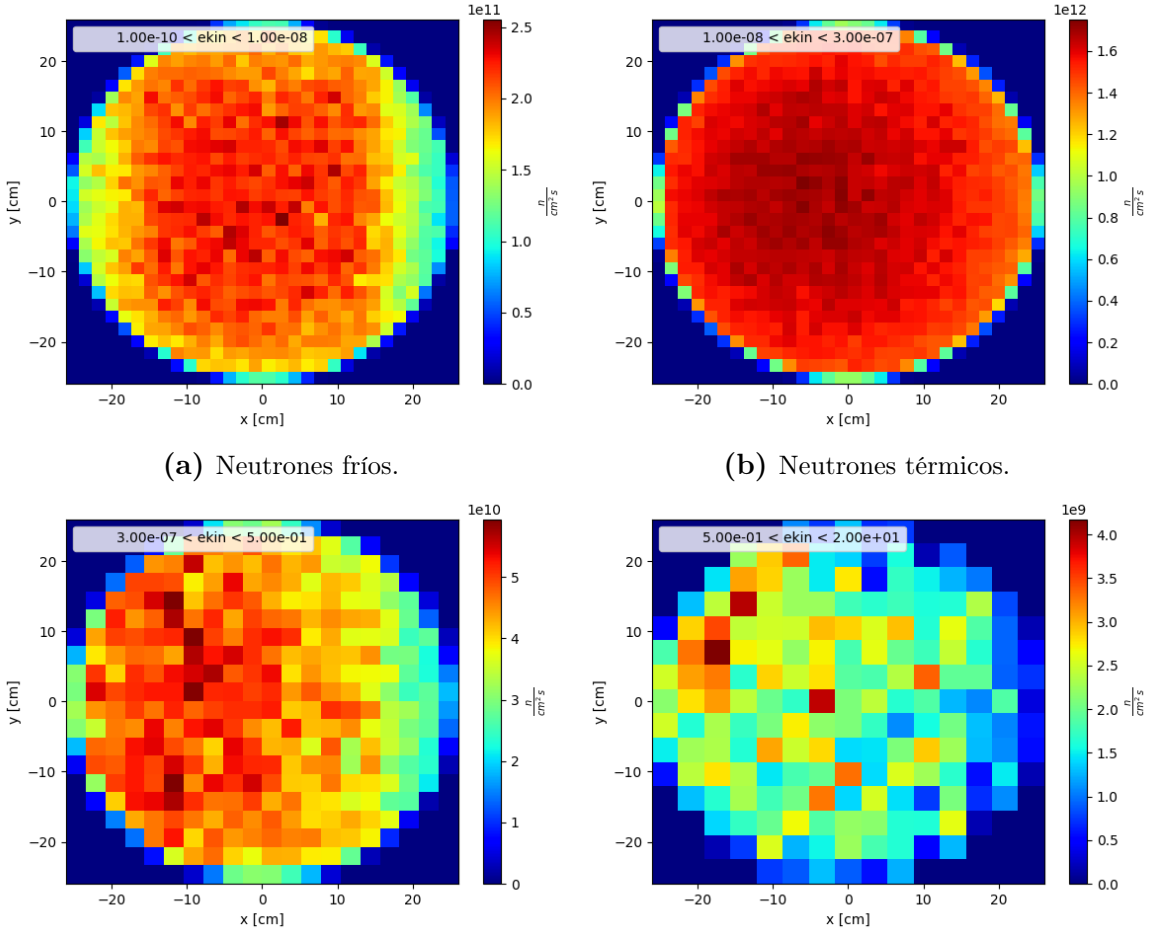
Figura 3.7: Ubicación de la fuente de *tracks* en la geometría empleada para su grabado en MCNP6.

se obtienen las siguientes corrientes medias:

$$J_n = 1,36 \cdot 10^{12} \ \frac{n}{cm^2 s} \tag{3.10}$$

$$J_\gamma = 7,42 \cdot 10^{11} \ \frac{f}{cm^2 s} \tag{3.11}$$

A continuación se presentan gráficos de la fuente de neutrones. Las principales observaciones que podrían realizarse coinciden con las realizadas en [2], donde se caracterizaron en detalle todos los haces del RA10, por lo que el análisis en este caso será breve. Para empezar, de las distribuciones de corriente en x e y se puede observar que los neutrones fríos se concentran en una región rectangular, lo cual se debe a que ésta es la forma de conducto de extracción que une la fuente fría con el tubo de vuelo. Por otra parte, los neutrones rápidos se concentran principalmente en la zona izquierda de la fuente, la más cercana al núcleo del reactor. Con respecto a la distribución energética, como era esperado, sólo los neutrones con μ cercano a 1 presentan el pico de neutrones fríos, ya que sólo éstos provienen de la fuente fría. Por último, en la distribución angular de neutrones fríos se observa que en la posición central el pico de corriente se concentra en $\mu = 1$, y al mover en la dirección x la macro zona considerada, éste se corre hacia menores valores de μ, lo cual se explica considerando que el pico se corresponde a los



(a) Neutrones fríos.

(b) Neutrones térmicos.

(c) Neutrones epitérmicos.

(d) Neutrones rápidos.

Figura 3.8: Corriente de neutrones fríos, térmicos, epitérmicos y rápidos, en función de las micro zonas de x e y.

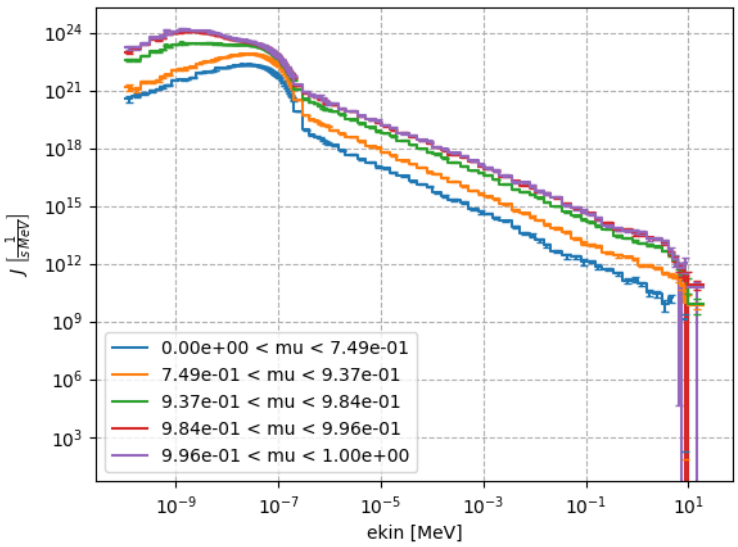
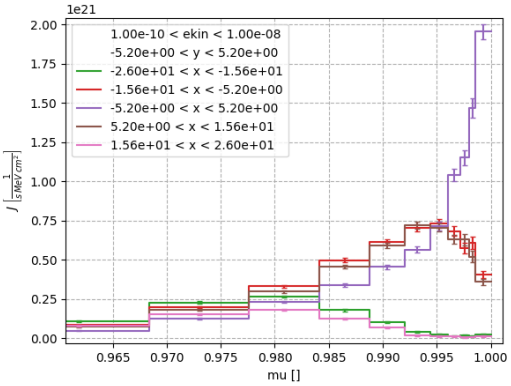
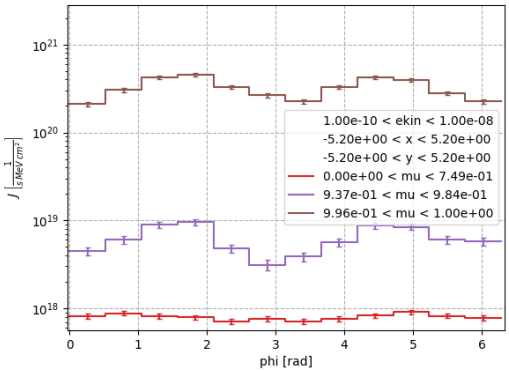


Figura 3.9: Corriente en función de micro grupos de energía, variando macro μ.



(a) Corriente de neutrones fríos en función de micro μ, variando x macro.



(b) Neutrones fríos en función de micro ϕ, variando macro μ.

Figura 3.10: Distribución angular de los neutrones fríos de fuente, tanto en μ como en ϕ.

neutrones que llegan desde la fuente fría, y cuanto más lejano al centro sea el punto considerado, más alejada del eje z será la dirección de vuelo de éstos. En la región central, la distribución en ϕ presenta dos picos en $\pi/2$ y $3\pi/2$, es decir hay una mayor cantidad de neutrones fríos viajando hacia arriba o hacia abajo que hacia los costados, lo cual se explica teniendo en cuenta que la fuente fría es más alta que ancha.

A continuación se presentan gráficos de la fuente de fotones. Éstos provienen de las reacciones (n, γ) *prompt* simuladas en la corrida de MCNP. Debido a la gran cantidad de decaimientos posibles, resulta difícil identificar precisamente el origen de los fotones. Sin embargo, en la Figura 3.9 se observa que las partículas γ provenientes de la fuente fría, es decir aquellas con μ cercano a 1, presentan un pico en la región de entre 0,2

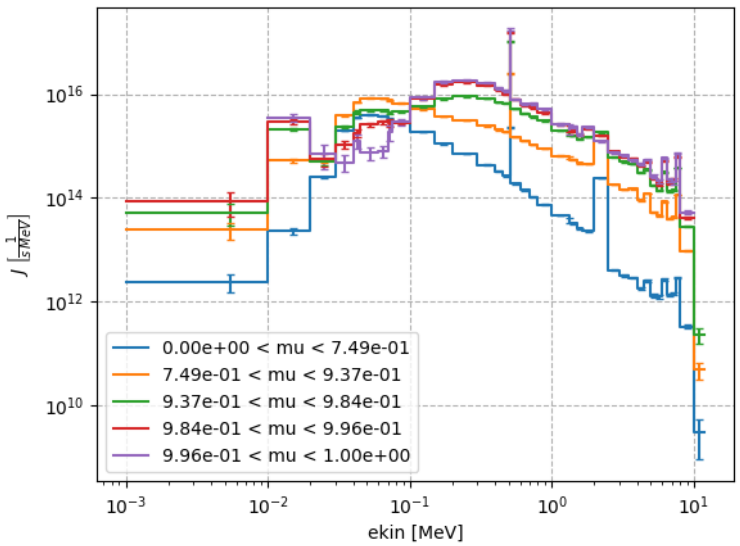
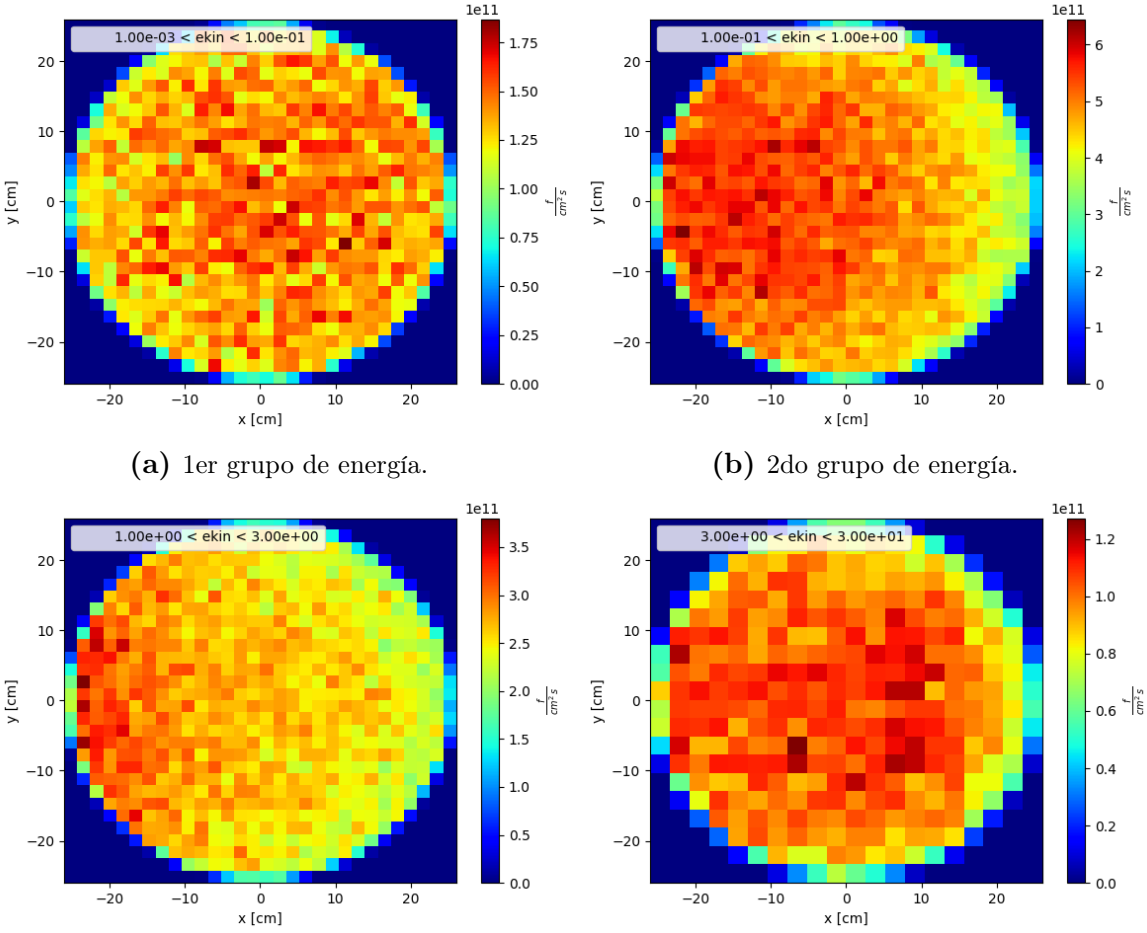


Figura 3.11: Corriente en función de micro grupos de energía, variando macro μ.



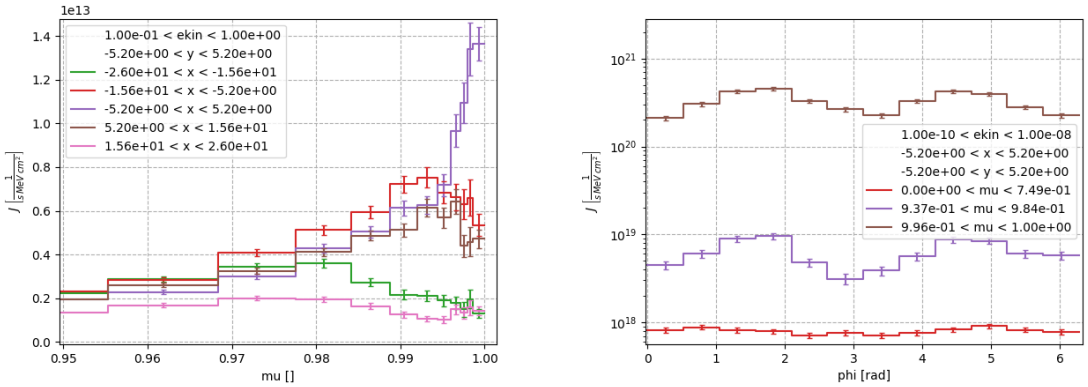
(a) 1er grupo de energía.

(b) 2do grupo de energía.

(c) 3er grupo de energía.

(d) 4to grupo de energía.

Figura 3.12: Corriente de fotones en función de las micro zonas, para los 4 grupos de energía.



(a) Corriente de fotones en función de micro μ, variando x macro.

(b) Neutrones fríos en función de micro ϕ, variando macro μ.

Figura 3.13: Distribución angular de los fotones de fuente, tanto en μ como en ϕ, para el 2do grupo de energía.

y 0,3 MeV, correspondiente al 2do macro grupo de energía. Esto es coherente con las distribuciones angulares que se observan en la Figura 3.13, para este mismo grupo energético, las cuales presentan los mismos patrones que los neutrones fríos, lo cual indica que se trata de fotones provenientes de la fuente fría.

3.5. Resultados de las simulaciones

En las siguientes subsecciones se presentan los resultados de las diferentes etapas simuladas, descritas en la Sección 3.3.

3.5.1. Discretizaciones empleadas

Para la elección de las discretizaciones en la entrada de GF1 se fijó el siguiente criterio: la corriente de neutrones a la salida, obtenida empleando una fuente de distribuciones a la entrada, debe ser mayor al 95 % que la obtenida sorteando los neutrones de la lista de *tracks*. No se considera una cota superior para dicho porcentaje ya que éste nunca resultó mayor al 100 %. Estas mismas discretizaciones, salvo la de energía, fueron también empleadas para la fuente de fotones a la entrada de la guía. En las demás fuentes de distribuciones se eligió cada conjunto de discretizaciones realizando ensayos y evaluando los resultados, balanceando en todos los casos el error estadístico, el cual aumenta cuanto más finas sean las grillas, con el error debido a la deformación de las curvas de distribución, mayor cuanto más gruesas sean éstas.

Para simplificar el proceso de inspección se fijaron ciertos criterios de antemano. Con respecto a la energía, en el caso de los neutrones se empleó una grilla fina de 138 grupos, la misma que la empleada por Fairhurst en [2], y una gruesa de 10 grupos. Inicialmente se intentó emplear sólo los siguientes 4 macro grupos: uno frío entre 0,1 meV y 10 meV, uno térmico entre 10 meV y 30 eV, uno epitérmico entre 30 eV y 500 keV, y uno rápido entre 500 keV y 20 MeV. Sin embargo, para lograr el criterio de una corriente a la salida mayor al 95 % de aquella obtenida con *tracks* se debió dividir en 4 subgrupos al grupo frío y en otros 4 al grupo térmico. Con respecto a la energía fotónica, se empleó la grilla fina denominada Vitamin B6, de 42 grupos, y una grilla gruesa de 4 grupos, con límites en 1 keV, 100 keV, 1 MeV, 3 MeV y 30 MeV. Por otra parte, para las discretizaciones de variables espaciales se emplearon grillas equiespaciadas, tanto en el caso de `x` e `y` en las fuentes de tipo `window` como en `z` y `mirror` en fuentes `guide`, y las 3 coordenadas `x`, `y`, `z` en fuentes `activ`. También se emplearon grillas equiespaciadas para la variable `phi` en todas las fuentes implementadas. Para la variable `mu`, en el caso de fuentes de tipo `window`, se debe emplear una grilla que concentre los grupos cerca de `mu=1`, ya que allí se da el pico de corriente. Por este motivo, se decidió emplear valores de corte μ_i tales que sus complementos $1 - \mu_i$ estén equiespaciados logarítmicamente.

Esto permite obtener valores de μ_i desde 0 hasta un número arbitrariamente cercano a 1, por lo que se debe agregar como límite del último *bin* el valor 1. Es decir que, con este esquema, no sólo se debe determinar la cantidad de grupos, sino también el valor del anteúltimo corte, lo cual afecta notablemente la grilla resultante. Para el caso de fuentes de tipo `guide`, el pico de corriente se da cerca de $\mu = 0$ (partículas "rasantes"), por lo que el esquema para los μ_i es análogo, pero tomando los complementos de 1 de los valores antes mencionados.

Considerando estos criterios, en cada tramo de la simulación se eligió la cantidad de grupos más adecuada para cada variable.

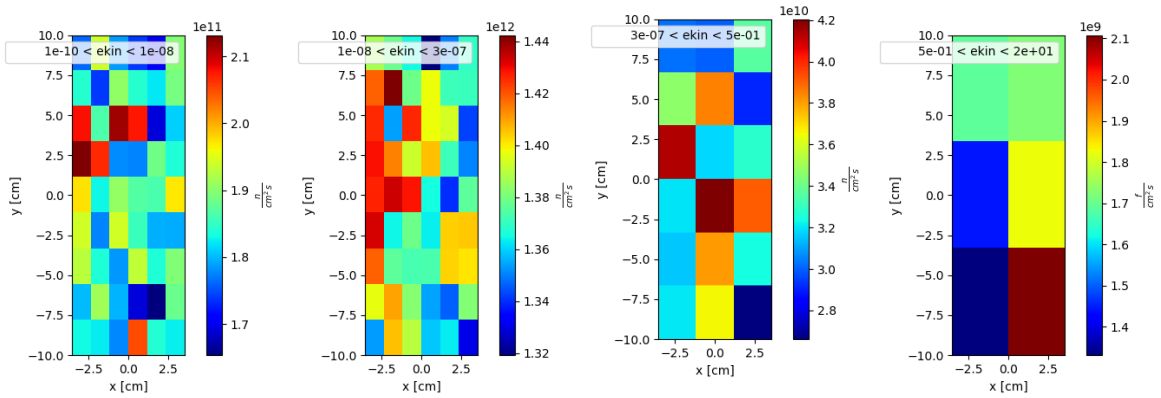
3.5.2. Tubo de vuelo

En el primer tramo de la simulación se sortearon las partículas directamente de las listas de *tracks*, mediante el código `Ptrac_source_difra.comp`. Si bien mediante esta técnica el error estadístico a la entrada de la guía resulta mayor que si se empleara una fuente de distribuciones, resulta mucho más sencillo lograr el objetivo de tener una corriente de neutrones mayor al 95 % del valor con *tracks* al final de la guía, y de todos modos la baja cantidad de partículas se limita a la entrada de la guía, ya que a partir de allí se se emplearán fuentes distribucionales. En dicha posición se registró una cantidad de $1{,}84 \cdot 10^5$ neutrones y $4{,}17 \cdot 10^4$ fotones, con lo cual las corrientes medias, promediadas en la sección de 7 por 20 cm, resultan:

$$J_n = 1,61 \cdot 10^{12} \tag{3.12}$$

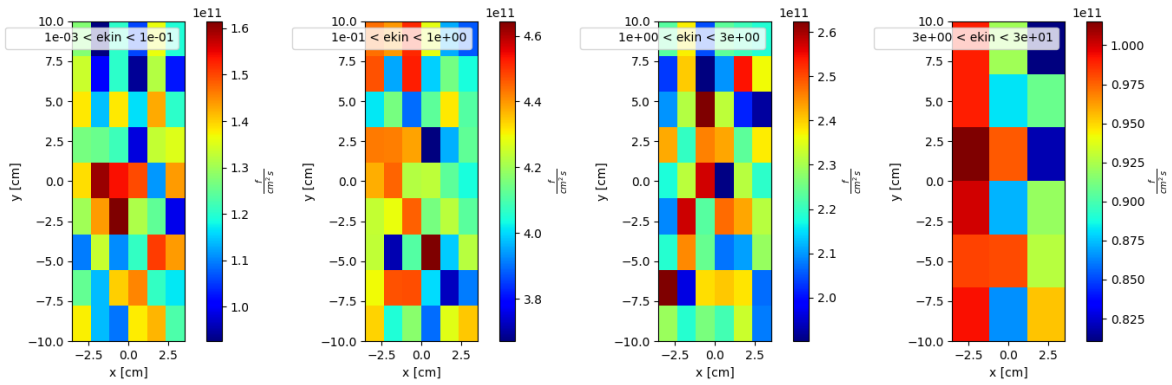
$$J_\gamma = 8,65 \cdot 10^{11} \tag{3.13}$$

En las figuras 3.14 y 3.15 se muestran las corrientes de neutrones y fotones en la



(a) Neutrones fríos. **(b)** Neutrones térmicos. **(c)** Neutrones epitérmicos. **(d)** Neutrones rápidos.

Figura 3.14: Corrientes de neutrones en la sección de entrada a la guía GF1, para los 4 rangos de energía.



(a) 1er grupo. **(b)** 2do grupo. **(c)** 3er grupo. **(d)** 4to grupo.

Figura 3.15: Corrientes de fotones en la sección de entrada a la guía GF1, para los 4 grupos de energía.

sección de entrada. En el caso de los neutrones, si bien se cuenta con 4 macro grupos de energía para el rango frío y 4 para el rango térmico, se integró la corriente en cada uno de estos intervalos. Las perturbaciones que se observan se atribuyen principalmente al error estadístico. Las distribuciones angulares y energéticas son muy similares a las que se observan en la fuente, ya que las partículas llegan desde allí sin interacción alguna.

3.5.3. Interior de la guía

Se realizaron dos corridas, una para neutrones y otra para fotones, sorteando en cada una 10^{10} partículas, y empleando las fuentes de distribuciones registradas en la etapa anterior. En ambos casos se emplearon tres fuentes tipo `guide`, dos rectas y una curva, de acuerdo a las dimensiones del haz (véase Tabla 3.1). Al final de la guía se empleó una fuente tipo `window`, de las dimensiones de la sección transversal, para registrar la distribución de neutrones saliente.

Gracias a los componentes empleados en McStas para modelar las guías, denominados `McStas2Distrib_guide.comp` y descritos anteriormente, se obtuvieron las distribuciones de neutrones y fotones salientes. En el caso de los fotones todos los escapes se dan en los tramos BC y D, sin que ninguno alcance el tramo E ni la salida. Esto se explica por la curvatura del tramo D, que impide que una partícula γ llegue al final de dicho tramo sin tocar alguno de los espejos. Como, desde luego, los fotones no son reflejados en éstos, escapan de la guía, siendo registrados por las fuentes de distribuciones.

Todas las corrientes medias en cada posición se observan en la Tabla 3.3, donde BC, D y E se refieren a los escapes a través de cada tramo de la guía. Se satisfacen las

siguientes relaciones, donde A_i refiere a cada área:

$$J_n^{salida} = 0,961 J_{n,tracks}^{salida} = 0,963 J_{n,Fairhurst}^{salida} \tag{3.14}$$

$$J_n^{BC} A_{BC} + J_n^{D} A_D + J_n^{E} A_E + J_n^{salida} A_{salida} = 0,997 J_n^{entrada} A_{entrada} \tag{3.15}$$

$$J_\gamma^{BC} A_{BC} + J_\gamma^{D} A_D + J_\gamma^{E} A_E + J_\gamma^{salida} A_{salida} = 0,997 J_\gamma^{entrada} A_{entrada} \tag{3.16}$$

La primera ecuación confirma que la corriente obtenida al emplear la fuente de distribuciones efectivamente reproduce la corriente a la salida de la guía, dentro de un 5 %. Las otras dos relaciones verifican el balance de cantidad de partículas, es decir que los neutrones y fotones que ingresan por la entrada o bien escapan por los tramos BC, D y E o bien llegan a la salida. La discrepancia del 0,3 % se debe a escapes en los huelgos de 0,1 mm empleados entre cada par de componentes en McStas, especialmente entre la entrada y el primer tramo de guía.

	entrada	BC	D	E	salida
Área $[cm^2]$	140	16200	109080	172800	140
$J_n \left[\frac{n}{cm^2 s}\right]$ (Fairhurst)					6.39E+09
$J_n \left[\frac{n}{cm^2 s}\right]$ (*tracks*)	1.61E+12				6.40E+09
$J_n \left[\frac{n}{cm^2 s}\right]$	1.61E+12	1.36E+10	2.20E+07	2.37E+06	6.15E+09
$J_p \left[\frac{f}{cm^2 s}\right]$	8.65E+11	7.41E+09	6.36E+06	0.00E+00	0.00E+00

Tabla 3.3: Resultados de las simulaciones del interior de la guía, en McStas.

En la Figura 3.16 se observa la corriente de neutrones escapando a través de la guía en función de la variable `z`. Como era de esperarse, inicialmente se observa una gran cantidad de escapes, debido a la componente rápida del flujo, así como de neutrones de alta divergencia. A medida que éstos escapan, quedan sólo aquellos de baja energía y ángulo, con lo cual la tasa de escapes disminuye. En la Figura 3.17 se observa el espectro energético de los neutrones registrados en los escapes, para distintos valores de `z`. Allí se observa precisamente el efecto de los espejos, distorsionando el espectro inicial, muy similar al observado en la fuente de *tracks*, en uno mucho más frío, eliminando todos los neutrones de más de 100 meV cerca del final de la guía. Se puede observar también que la longitud de la guía no es suficiente para que las fugas lleguen a ser nulas, sino que a pesar del enfriamiento del espectro se siguen registrando escapes hasta el final de ésta. Para mayor detalle de la distribución espacial de los escapes, en la Figura 3.18 se observa la corriente saliente en función de las variables `z` y `mirror`. Se puede pensar a la imagen como si los cuatro espejos hubieran sido desplegados, de modo de poder verlos a todos en un mismo plano. En este caso, además de observarse la atenuación de la corriente al crecer `z`, se observa la mayor densidad de escapes en el espejo convexo, lo cual se explica por el mayor ángulo con el cual inciden los neutrones. Por último,

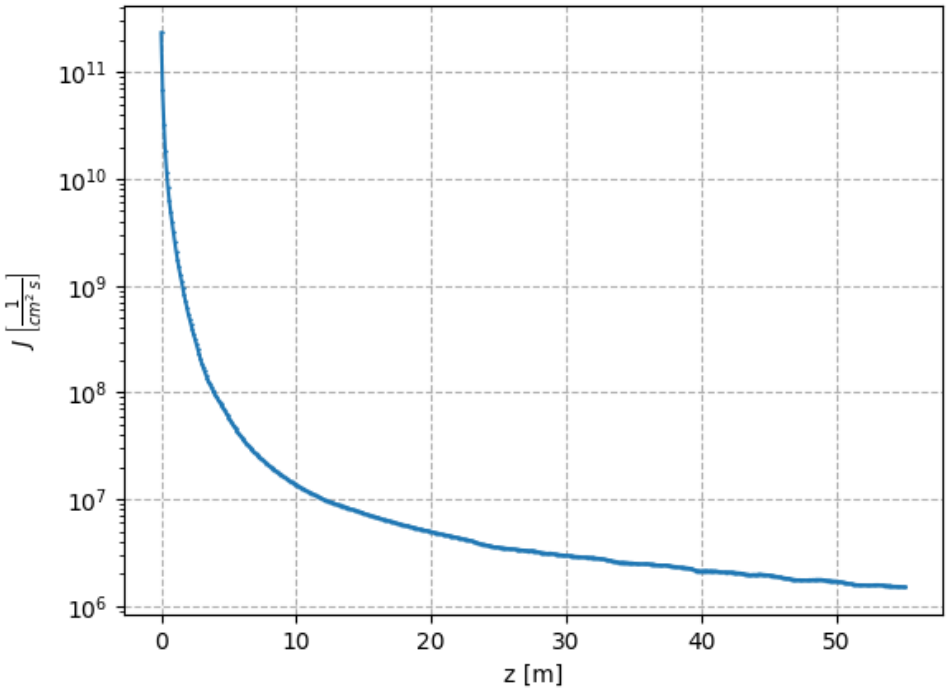


Figura 3.16: Corriente de neutrones escapando de la guía, promediada en el perímetro de cada sección, en función de z.

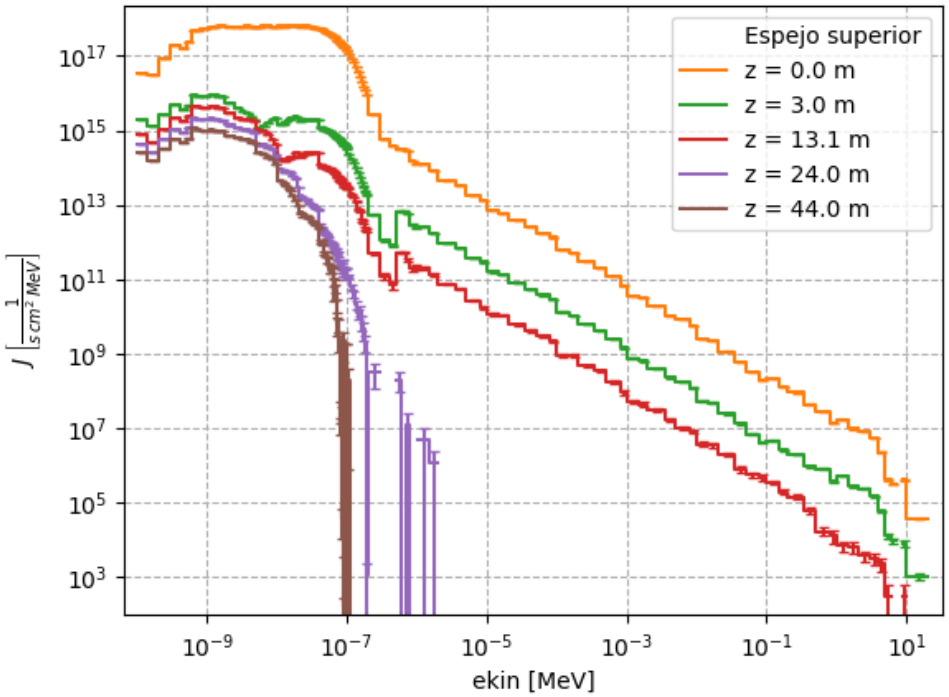


Figura 3.17: Espectro energético de los neutrones escapando de la guía, para varios valores de de z.

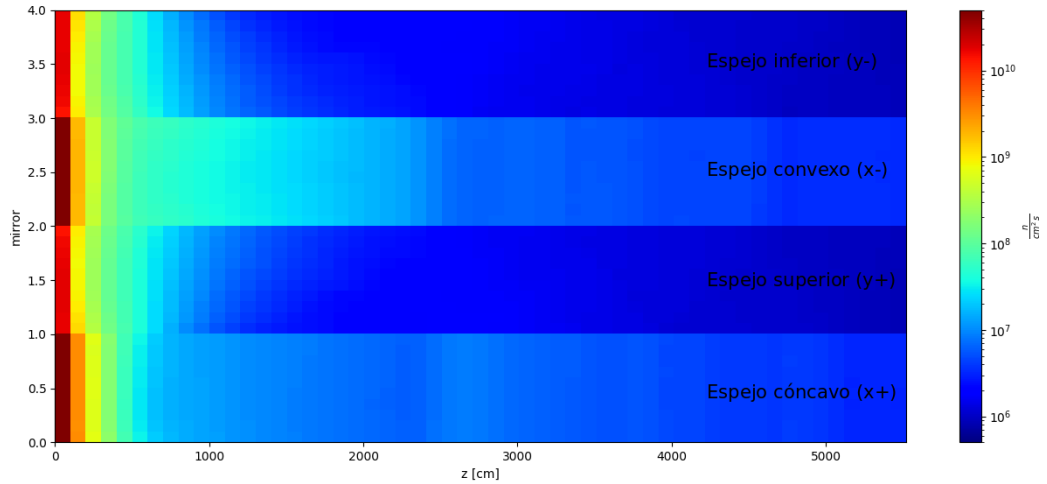
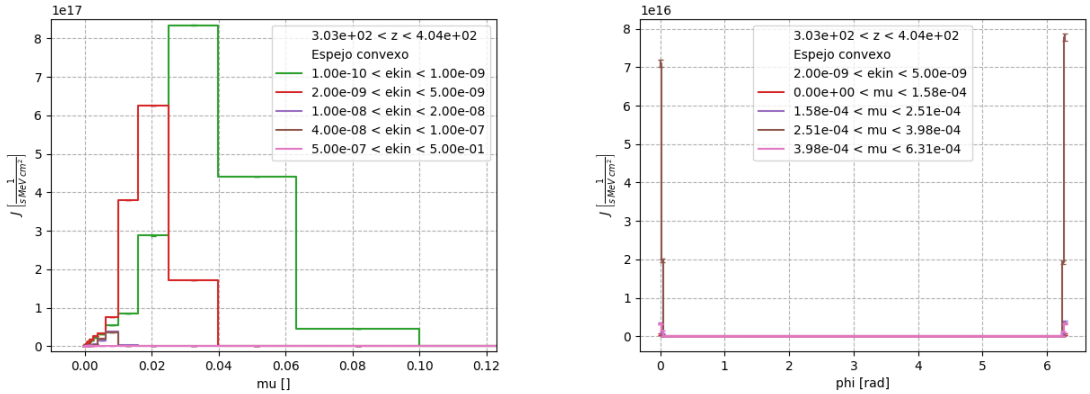


Figura 3.18: Corriente de neutrones escapando de la guía, en función de z y la variable `mirror`.



(a) Corriente en función de μ, para varias energías.

(b) Corriente en función de ϕ, para varios valores de μ.

Figura 3.19: Distribución angular de la corriente de neutrones escapando de la guía.

en la Figura 3.19 se observa la distribución angular de los escapes, en la región final del tramo BC. Como se esperaba, el pico está cerca de `mu=0`, ya que los neutrones con alta divergencia ya han escapado en los primeros centímetros de la guía, pero tampoco exactamente en dicho valor, ya que aquellos con muy baja divergencia son reflejados. Desde luego, todos los escapes se dan en la región cercana a `phi=0`, es decir en la dirección del eje `z`.

En las siguientes figuras se observan los resultados análogos a los anteriores para el caso de los fotones. Con respecto a la corriente de escapes en función de `z`, observada en la Figura 3.20, la atenuación se vuelve más brusca hacia el final del tramo D, por las razones ya descritas. La distribución de corriente en los cuatro espejos desplegados, de la Figura 3.21, presenta, al igual que para los neutrones, mayores valores en el espejo convexo, sobre el cual inciden los fotones con baja divergencia. En este caso en el espejo cóncavo rápidamente se anula la corriente, por lo mismos motivos por los cuales

ningún fotón llega al tramo E. La distribución angular es completamente análoga a la de neutrones, por lo que no se muestran sus gráficos.

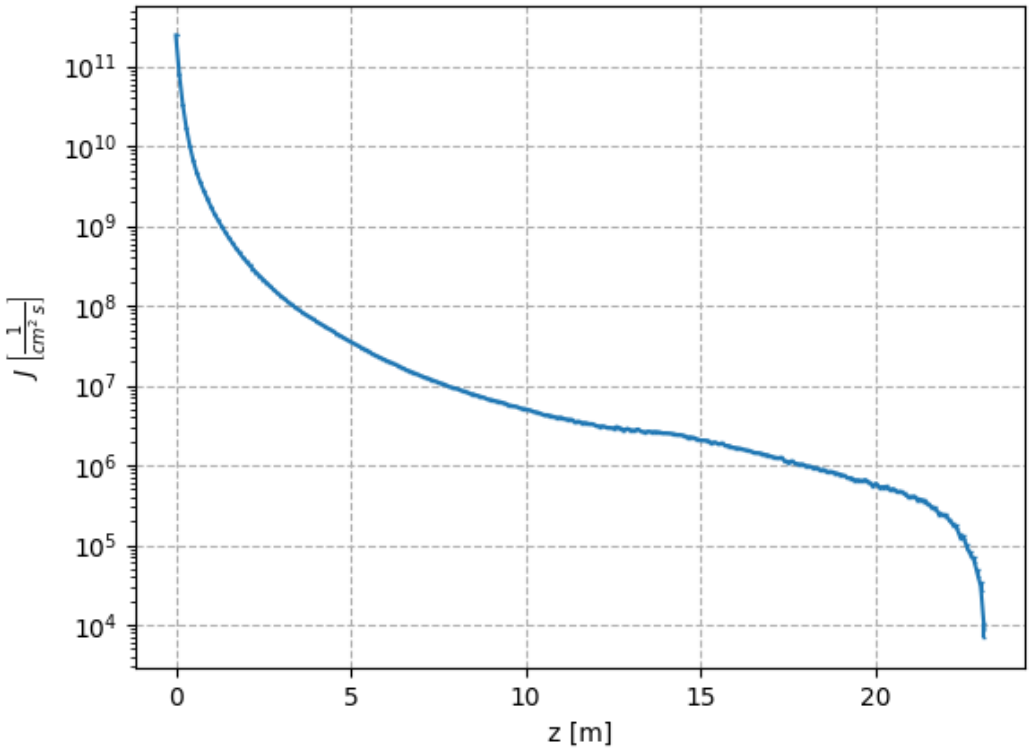


Figura 3.20: Corriente de fotones escapando de la guía, promediada en el perímetro de cada sección, en función de z.

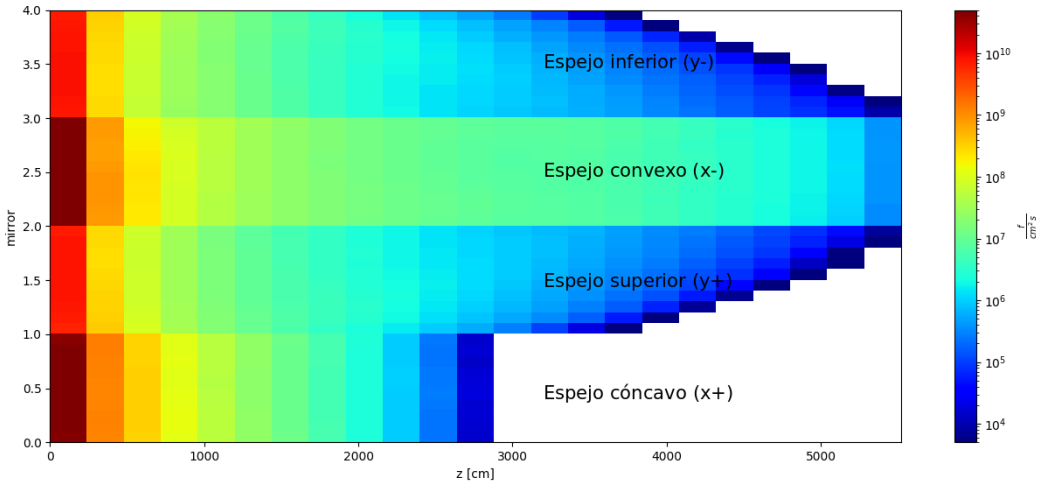
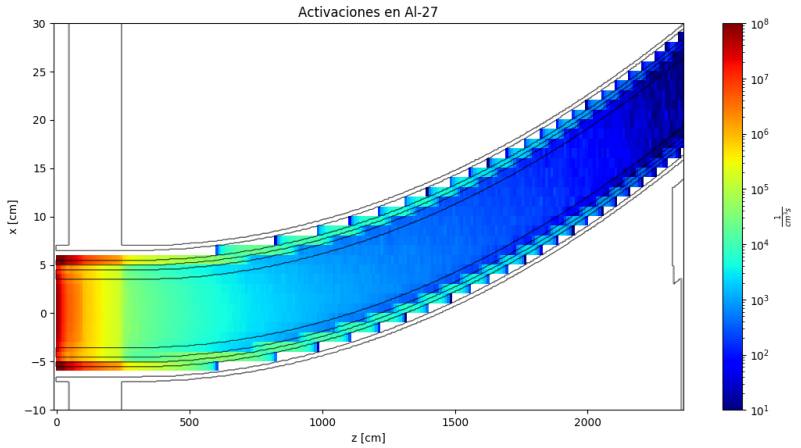


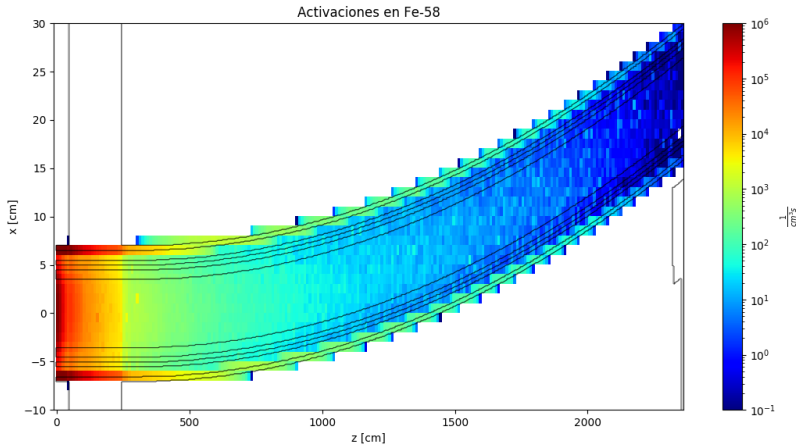
Figura 3.21: Corriente de fotones escapando de la guía, en función de z y la variable `mirror`.

3.5.4. Búnker de guías

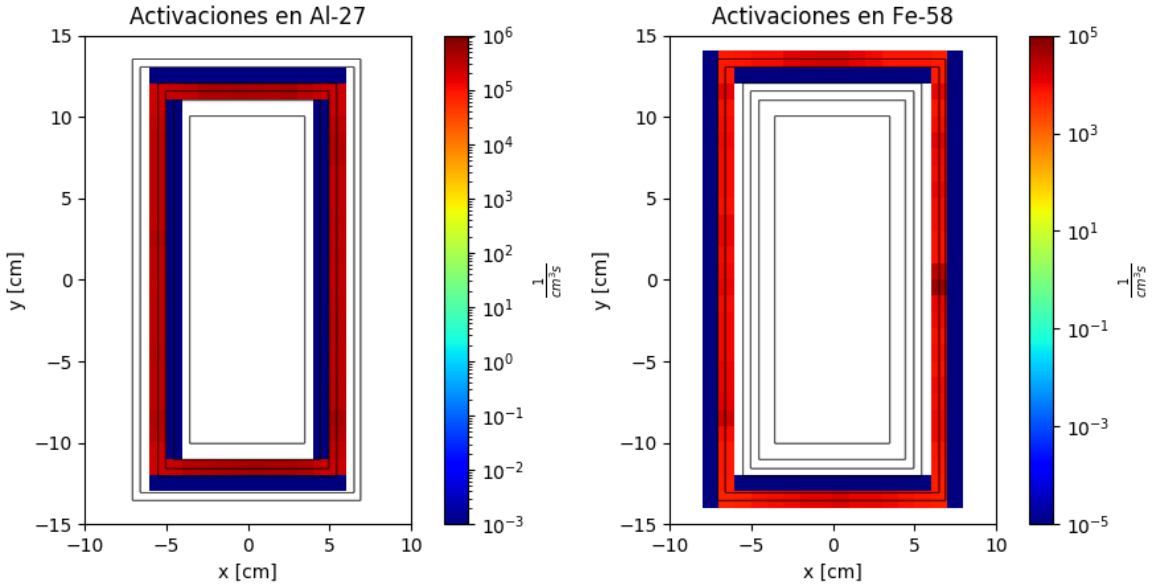
Con la corriente de neutrones saliendo de los tres tramos de la guía se efectuó una simulación en Tripoli, en la cual se registró un *tally* de dosis ambiental, y además otros



(a) Activaciones de Al-27 promediadas en la dirección y.



(b) Activaciones de Fe-58 promediadas en la dirección y.



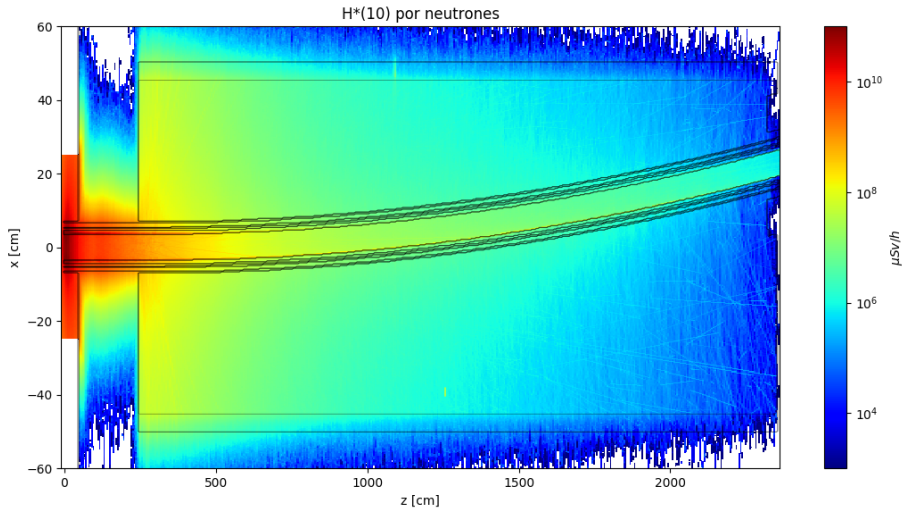
(c) Activaciones de Al-27 en un corte transversal. **(d)** Activaciones de Fe-58 en un corte transversal.

Figura 3.22: Mapas de los *tallies* de activación registrados, ubicados principalmente en el sistema de alineación (aluminio) y el sistema de vacío (hierro).

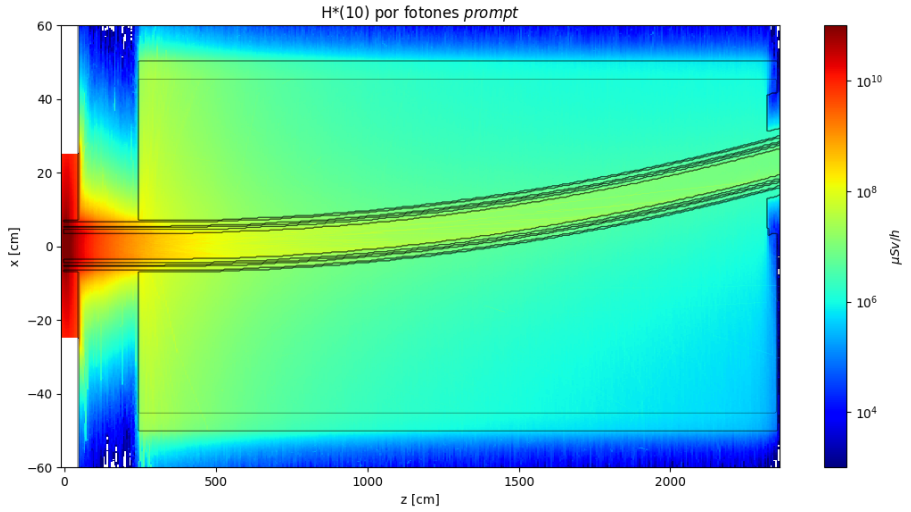
dos de activación en Al-27 (sistema de alineación) y Fe-58 (sistema de vacío). En dicha simulación son generados por Tripoli fotones de reacciones (n, γ) *prompt*, por lo que también se registró la dosis debida a éstos. Posteriormente, los *tallies* volumétricos de activación fueron convertidos a fuentes de distribuciones de tipo `activ`, con las cuales se realizó una nueva simulación, registrando la dosis por fotones de activación. Por otra parte, se realizó una corrida empleando como fuentes las distribuciones de fotones registradas en la etapa anterior, registrando en este caso únicamente dosis ambiental. Por último, en todas las simulaciones se registraron las listas de *tracks* de partículas que entren a las paredes del búnker, manteniendo por separado los neutrones, los fotones de fuentes, los *prompt*, y aquellos de activación. En los sorteos de neutrones y fotones de fuente, implementados con el código `Distrib2Tripoli.c`, se utilizó la técnica de *source biasing* para favorecer la producción de partículas en el tramo D, el más significativo para esta etapa de la simulación. En cada una de las tres simulaciones se produjeron 10^9 partículas.

En la Figura 3.22 se muestran imágenes de los mapas de activación registrados. Los mismos consisten en *tallies* de reacciones de captura neutrónica en Al-27 y Fe-58. Se consideraron únicamente estos dos isótopos pues suelen ser los que aportan la mayor contribución a la radiación por decaimientos.

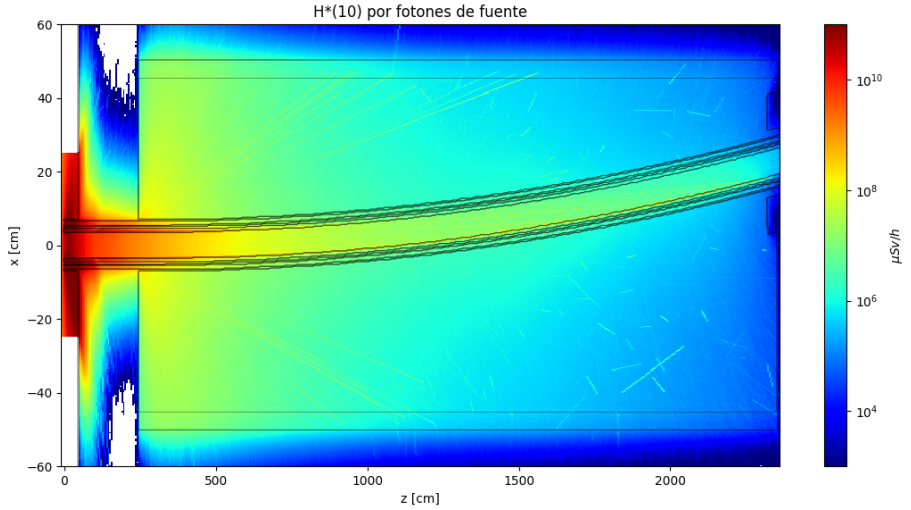
En la Figura 3.23 se muestran los mapas de dosis registrados por neutrones, fotones *prompt*, fotones de fuente y fotones de activación, en un corte horizontal del búnker. Se puede observar, como era de esperarse, que la zona más comprometida del búnker es la región más cercana al reactor. Para observarla en detalle, en la Figura 3.24 se muestran las dosis por las mismas cuatro partículas, pero en este caso en una sección transversal de la guía, promediada entre los planos $z = 350$ cm y $z = 450$ cm.



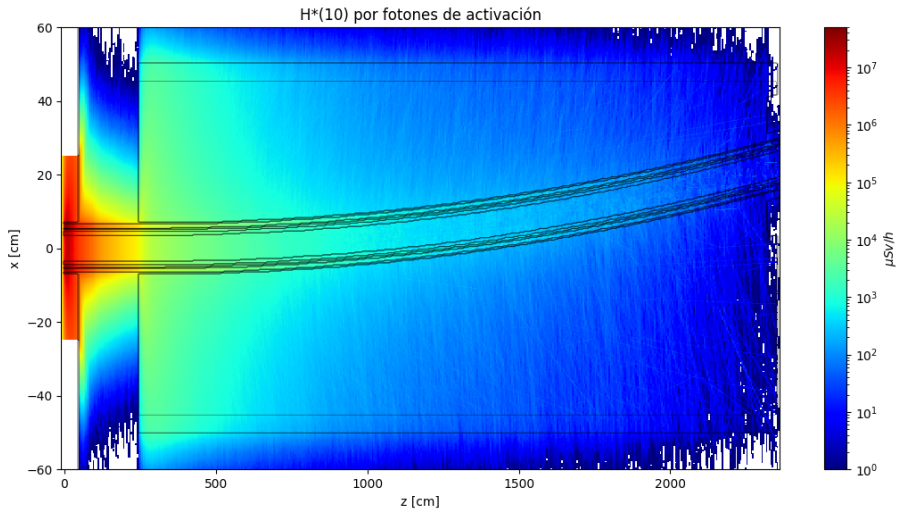
(a) Neutrones.



(b) Fotones *prompt*.

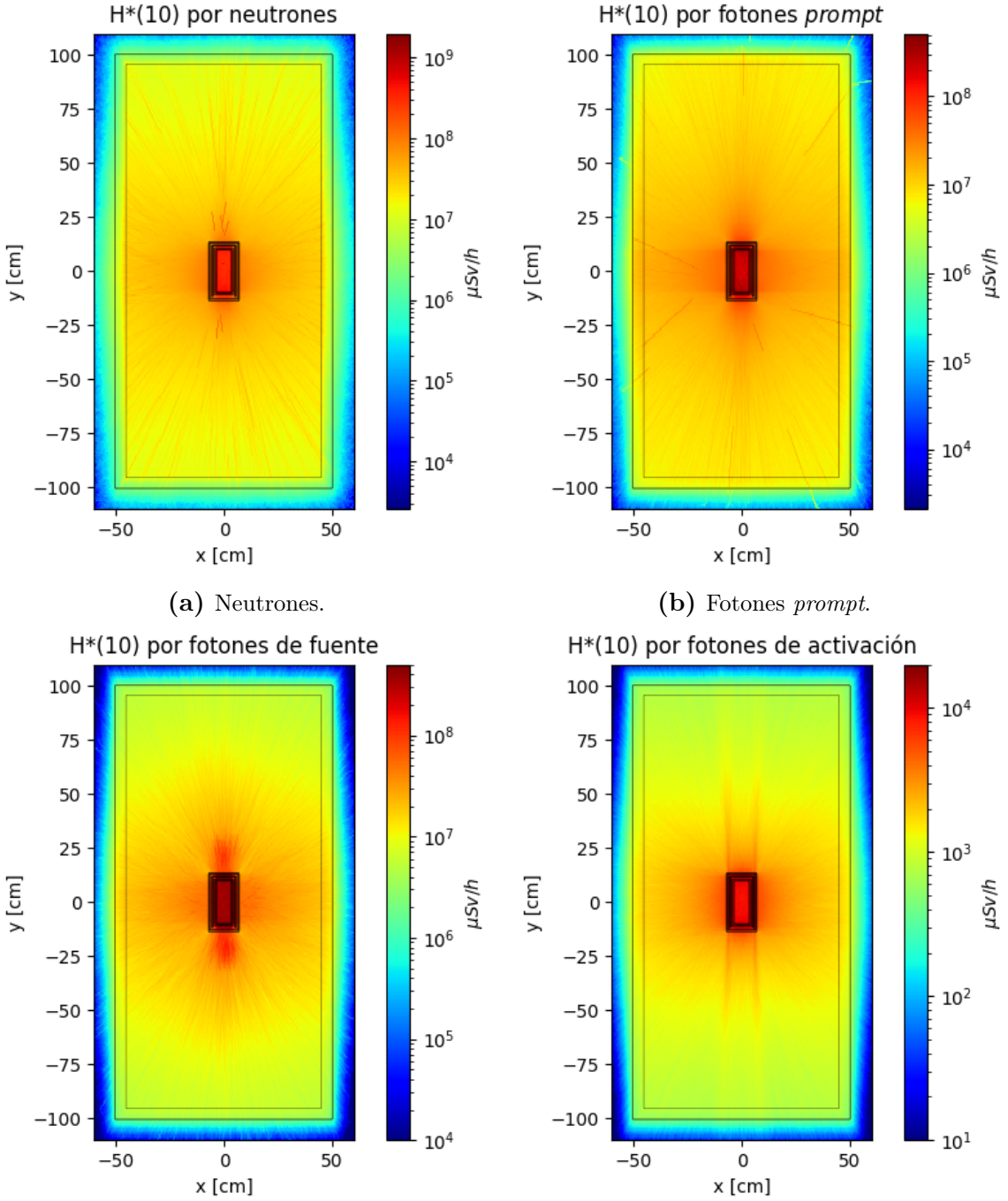


(c) Fotones de fuente.



(d) Fotones de activación.

Figura 3.23: Dosis ambiental en un corte horizontal del búnker de guías.



(a) Neutrones.

(b) Fotones *prompt*.

(c) Fotones de fuente.

(d) Fotones de activación.

Figura 3.24: Dosis ambiental en un corte transversal del búnker de guías, promediada entre en z entre los valores de 350 cm y 450 cm.

Para estudiar el impacto de no incluir electrones y positrones en las simulaciones, es decir no considerar los fotones de frenamiento (*bremsstrahlung*), se realizó una simulación de la misma región considerada en los últimos casos con dichas partículas. Se emplearon los fotones de fuente para la producción de partículas, y se observó que el tiempo de cálculo por partícula simulada fue 4,5 veces mayor, lo cual obligó a que en la corrida con electrones y positrones se produjeran 10 veces menos partículas que en

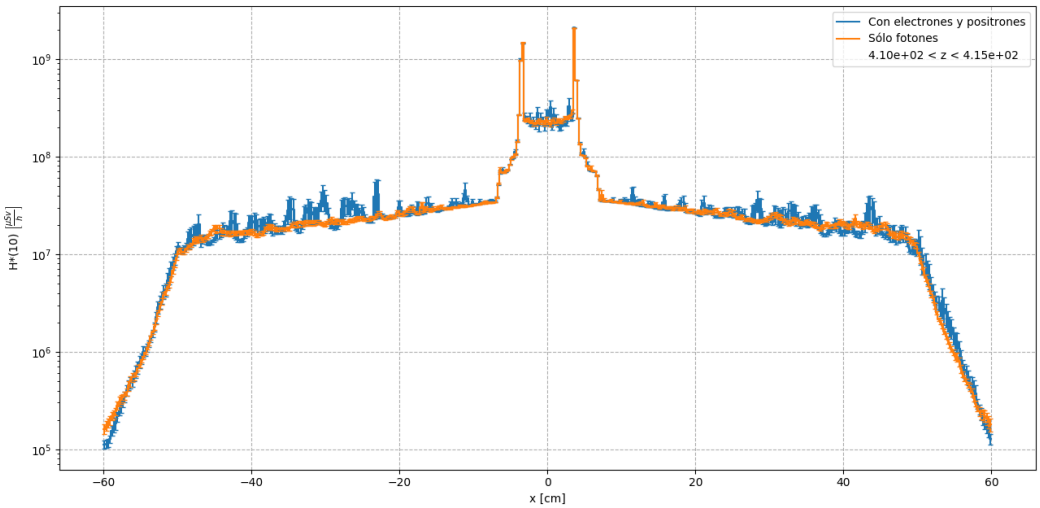


Figura 3.25: Comparación entre la dosis obtenida al utilizar fotones, electrones y positrones, y aquella empleando sólo fotones.

la corrida análoga sólo con fotones. En la Figura 3.25 se compara la dosis registrada en ambas simulaciones en un corte horizontal a $y = 0$ constante. Se observa que en la zona de los sistemas de la guía, así como dentro del búnker no hay diferencia apreciable. Al penetrar la pared del búnker, la diferencia es leve, siendo mayor la dosis registrada sólo con fotones. De este resultado se concluyó que emplear únicamente neutrones y fotones en las simulaciones, además de reducir notablemente los tiempos de cálculo, no tiene grandes implicancias en los resultados obtenidos, y las diferencias posibles tenderán a sobrestimar la dosis, lo cual hace que, en este sentido, el estudio sea conservativo.

Para su uso en la etapa siguiente, las listas de *tracks* registradas en las paredes

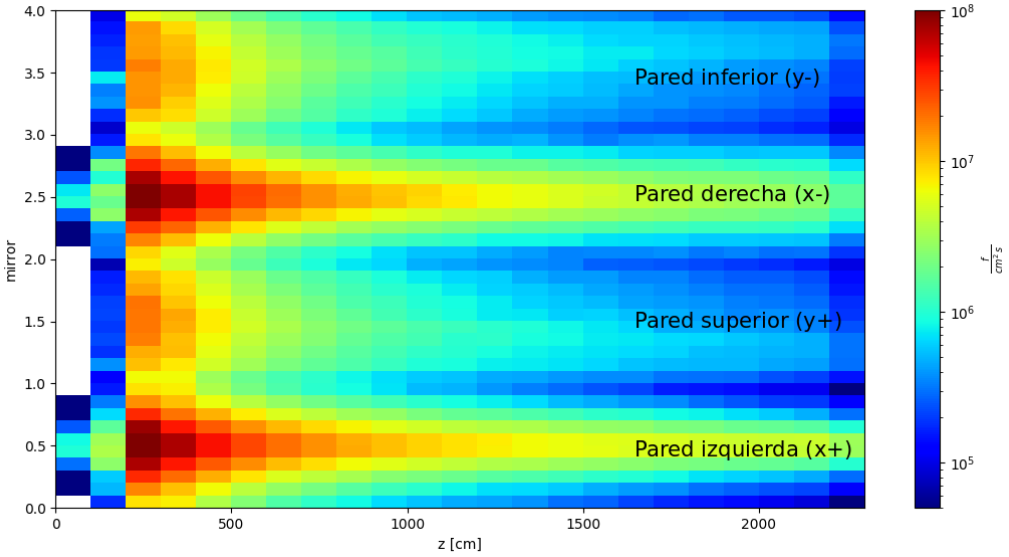


Figura 3.26: Corriente de fotones *prompt* entrando a las paredes laterales del búnker de guías, en función de z y la variable `mirror`.

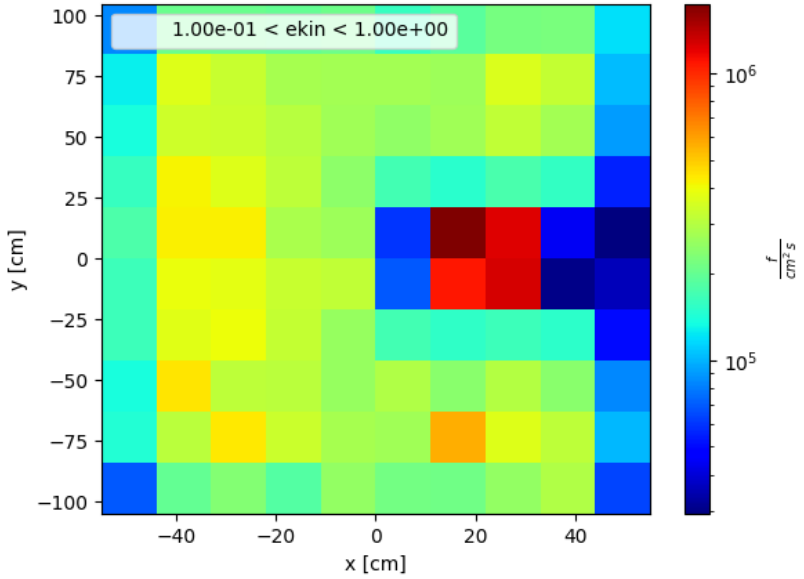


Figura 3.27: Corriente de fotones de fuente entrando a la pared frontal del búnker de guías, en función de x e y.

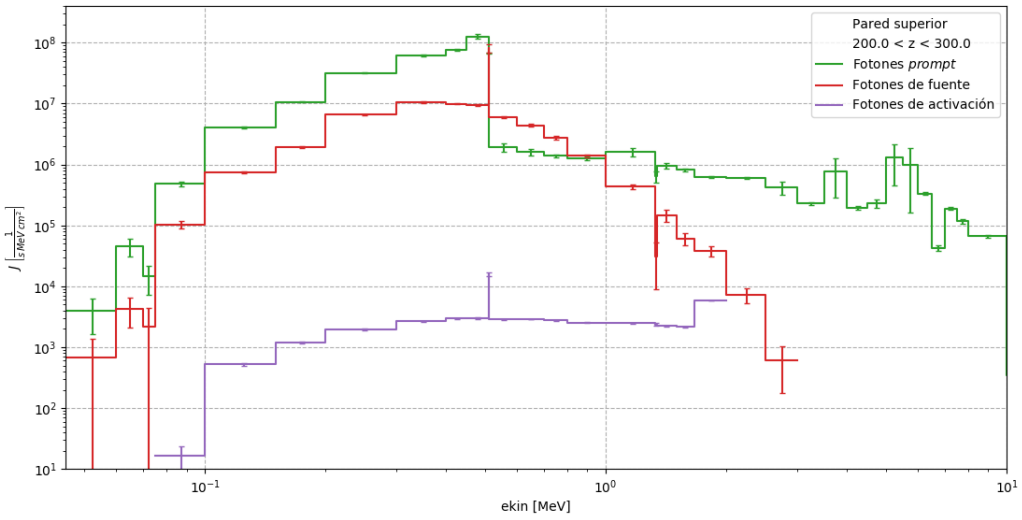


Figura 3.28: Comparación de los espectros energéticos de fotones de fuente, de reacciones *prompt* y de activación.

del búnker fueron convertidas a fuentes de distribuciones. A modo de ejemplo, en la Figura 3.26 se muestra la corriente de fotones *prompt* a través de las cuatro paredes laterales (incluyendo la superior e inferior) del búnker, desplegadas del mismo modo que en gráficos anteriores de las paredes de la guía. Imágenes similares se observan para el caso de neutrones, fotones de fuente y de activación. En la Figura 3.27 se muestra la corriente de fotones de fuente en la pared frontal, en función de x e y. Por último, en la Figura 3.28 se comparan los espectros energéticos de los tres tipos de fotones simulados, registrados al atravesar la pared superior (techo) del búnker.

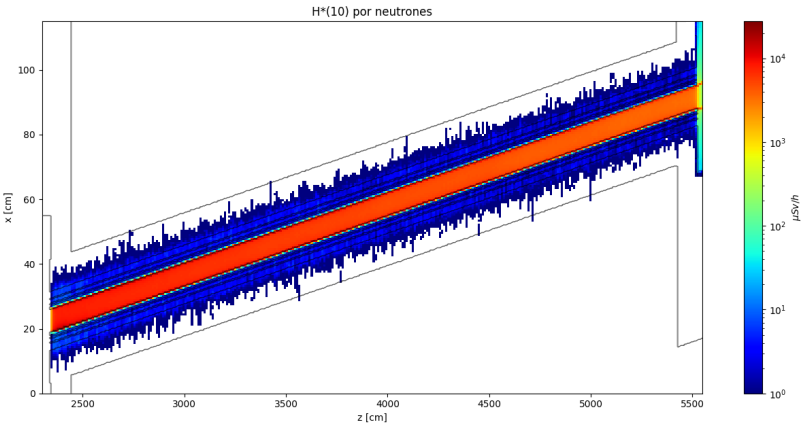
3.5.5. *Hall* de guías

Como se mencionó anteriormente, esta etapa de la simulación se realizó en dos partes: una utilizando las corrientes ingresando a las paredes del búnker registradas en la etapa anterior, y otra empleando los escapes de neutrones en el tramo E de la guía registrados en McStas. La simulación del blindaje del búnker consistió en tres corridas, una para neutrones y fotones *prompt*, otra para fotones de fuente y otra para fotones de activación. No se consideró la activación en materiales del hormigón pesado del búnker, como el hierro, por considerárselo despreciable. En la simulación del blindaje exterior del tramo E sí se calcularon *tallies* de activación y se corrieron dichos fotones posteriormente. Los resultados de cada etapa se estudiaron por separado, pero debido a que los campos de dosis de cada una se solapan parcialmente, especialmente en la región del blindaje exterior cercana al búnker, finalmente se sumaron las dosis en los resultados finales.

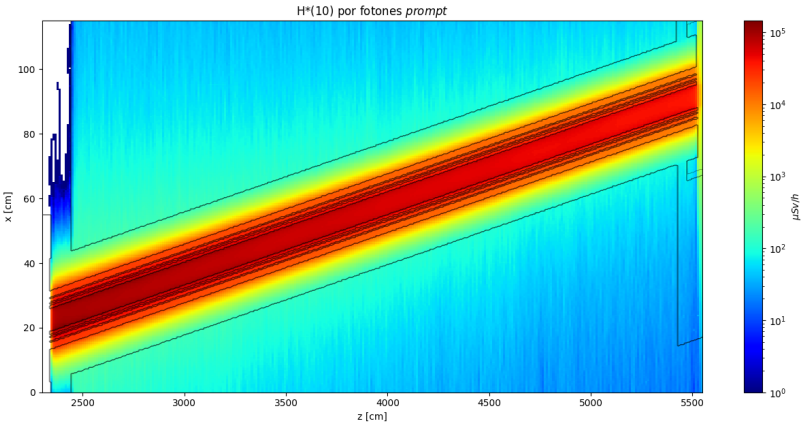
Blindaje exterior

En la Figura 3.29 se observan los *tallies* de dosis por neutrones y fotones *prompt* registrados en un plano horizontal del blindaje exterior. Se pueden observar tres puntos principales: primero, que la zona más comprometida es aquella más cercana al búnker, como era esperado; segundo, que la componente predominante de la dosis son los fotones *prompt*; por último, que la dosis fuera del blindaje está notablemente por encima de los 3 µSv/h, es decir que el blindaje es insuficiente. En la Figura 3.30 se observa en mayor detalle el mapa de dosis en una sección transversal de la región más comprometida, promediada entre $z = 2480$ cm y $z = 2520$ cm. La cantidad de partículas producidas fue de 10^9.

Para estimar el aumento requerido en el espesor de la pared, en la Figura 3.31 se observa el perfil de dosis en el centro de la guía, a lo largo del eje y, en escala logarítmica. Si se extrapola linealmente la curva de dosis, es decir asumiendo una



(a) Neutrones.



(b) Fotones *prompt*.

Figura 3.29: Primer resultado de dosis por fotones *prompt* en un plano horizontal el blindaje exterior.

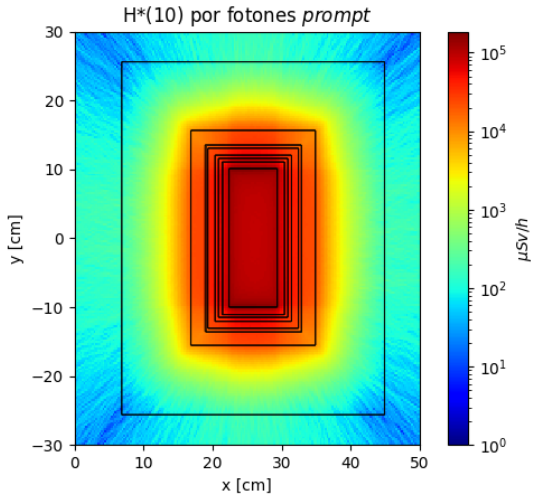


Figura 3.30: Primer resultado de dosis por fotones *prompt* en un plano transversal el blindaje exterior.

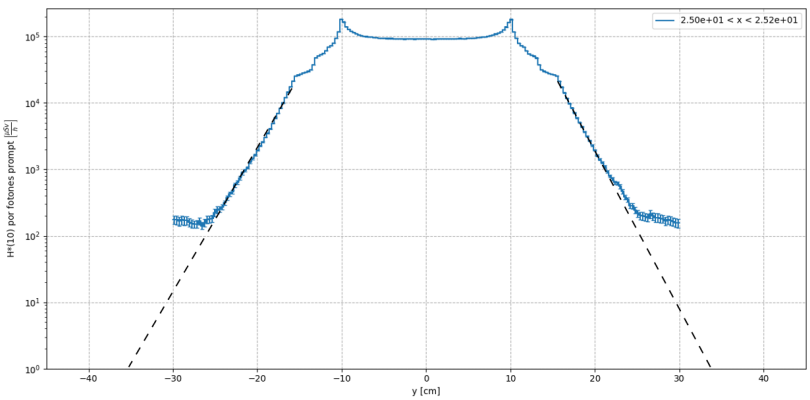


Figura 3.31: Perfil de dosis a lo largo del eje y, con la recta de extrapolación.

atenuación exponencial, se obtiene que se requieren aproximadamente 10 cm más de hormigón pesado. Por seguridad, y considerando que la atenuación de un campo con múltiples ángulos de propagación no es realmente exponencial, se agregaron 15 cm al blindaje, y se repitió la simulación, nuevamente con 10^9 partículas.

En la Figura 3.32 se observa la dosis en el mismo corte transversal mostrado anteriormente, y en la Figura 3.33 se muestra el perfil en el eje y. Se observa que la curva pierde su tendencia exponencial, haciendo que la dosis estimada mediante la curva de extrapolación subestime la dosis. El resultado es que, si bien la estadística no alcanza para tener una curva de nivel a 3 µSv/h bien definida, no resulta seguro que la dosis fuera del blindaje este por debajo de este valor, es decir que el blindaje nuevamente resulta insuficiente. Por este motivo se decidió emplear una capa de plomo recubriendo la superficie exterior del blindaje. Se empleó Pb natural puro, con una densidad de 11,35 g/cm^3, y un espesor de 7 cm.

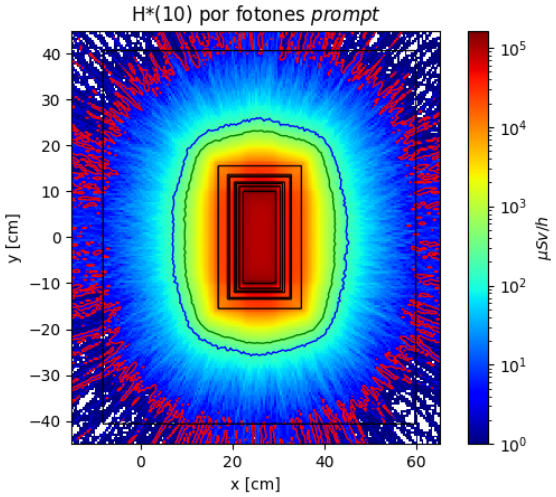


Figura 3.32: Segundo resultado de dosis por fotones *prompt* en un plano transversal el blindaje exterior. En rojo la curva de 3 µSv/h, en azul la de 200 µSv/h y en verde la de 500 µSv/h.

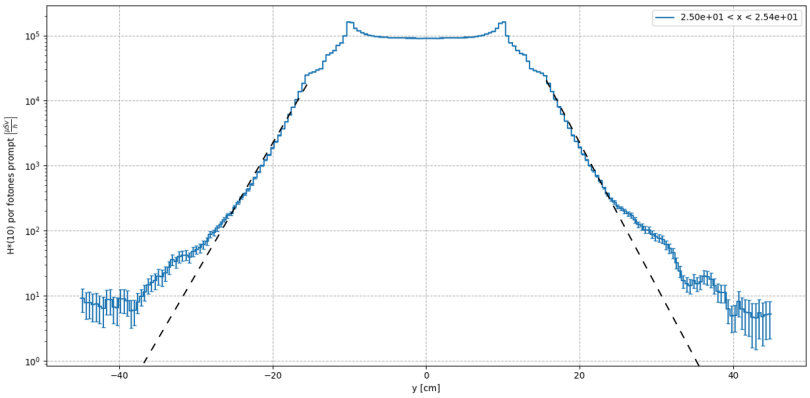
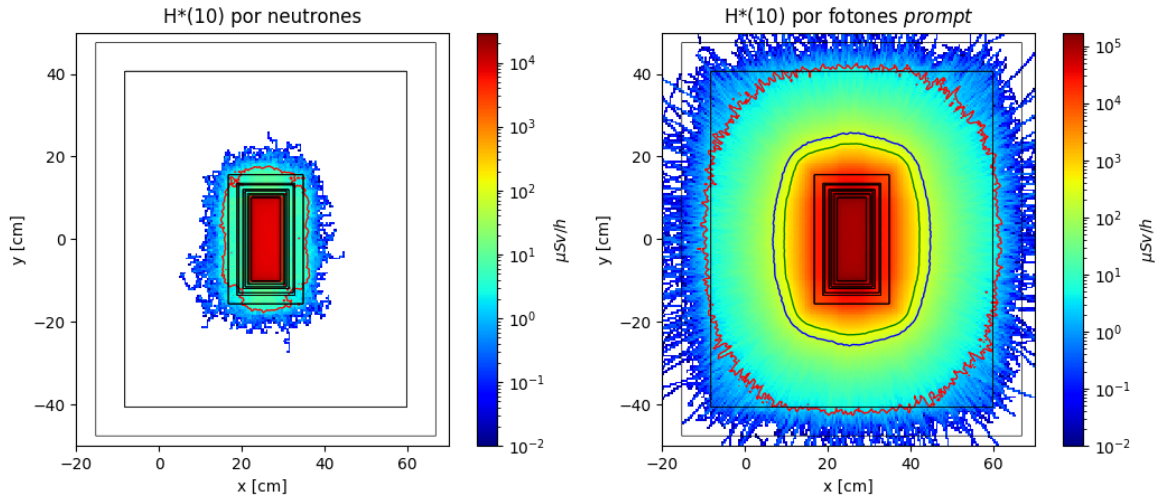


Figura 3.33: Perfil de dosis a lo largo del eje y, con la recta de extrapolación utilizada anteriormente.

En la Figura 3.34 se observan los mapas de dosis por neutrones y por fotones *prompt* en el corte transversal empleado anteriormente, para el diseño final de blindaje exterior, y una corrida de 10^{10} partículas. Se observa que en ambos casos las curvas de 3 µSv/h están claramente por dentro del blindaje. En particular, la dosis por neutrones resulta ser extremadamente baja luego del vidrio borado de los espejos, debido a que en este tramo de la guía la mayoría de los neutrones son térmicos, con lo cual la sección eficaz de absorción del boro es muy grande.

En esta etapa también se registraron mapas de activación en Al-27 y Fe-58, análogamente a lo realizado en el interior del búnker. En la Figura 3.35 se observa el mapa de dosis registrado en la sección transversal, para una corrida de 10^7 fotones. En este caso esta relativamente pequeña cantidad de producciones resultó suficiente para ob-



(a) Neutrones. **(b)** Fotones *prompt*.

Figura 3.34: Dosis ambiental en un corte transversal del blindaje exterior, para el diseño final. En rojo la curva de 3 µSv/h, en azul la de 200 µSv/h y en verde la de 500 µSv/h.

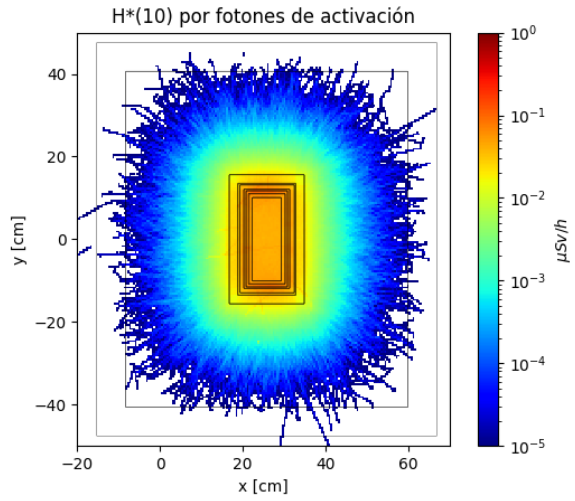


Figura 3.35: Dosis ambiental por fotones de activación en un corte transversal del blindaje exterior, para el diseño final.

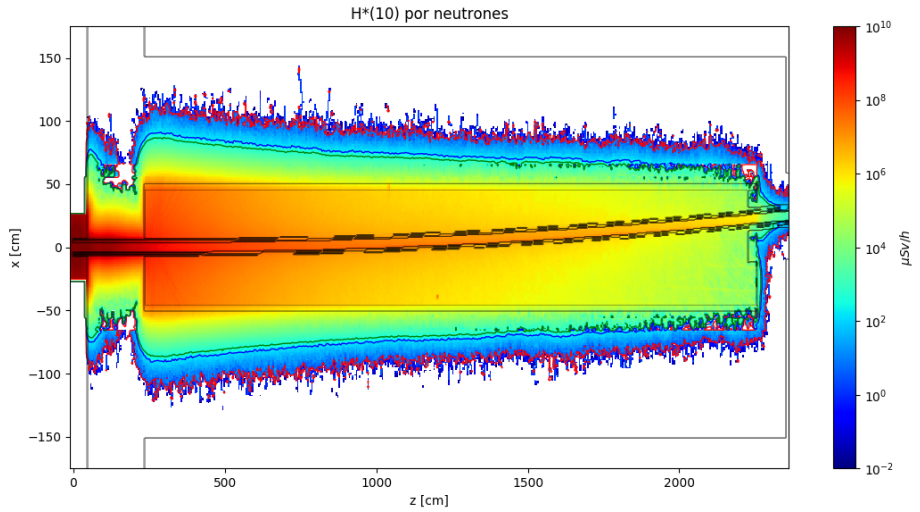
servar la baja tasa de dosis que producen en esta región los fotones de activación, que en ningún punto supera el valor de 1 µSv/h.

Blindaje del búnker de guías

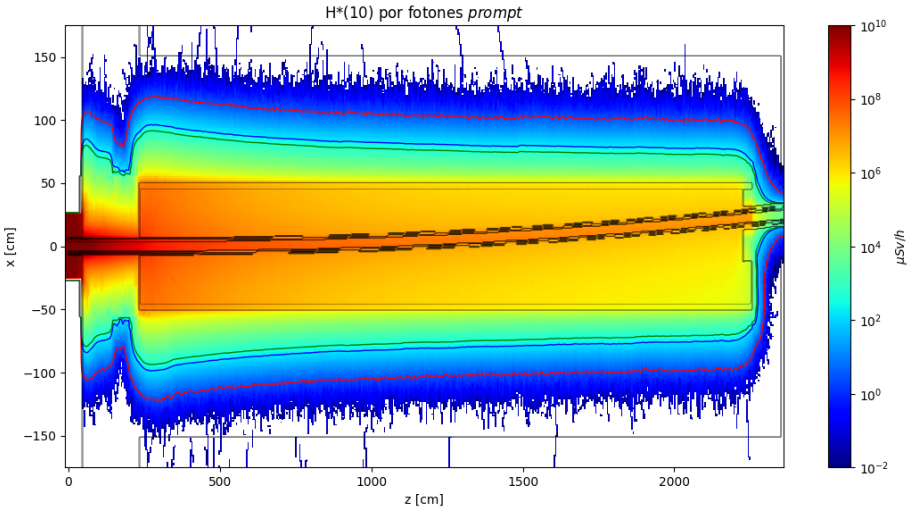
En la Figura 3.36 se observan los mapas de dosis registrados en un corte horizontal del *hall* de guías, mientras que en la Figura 3.37 se muestran aquellos obtenidos en un corte horizontal del mismo, promediado en z entre 350 cm y 450 cm. En las imágenes se superpusieron los resultados del interior del búnker en la región correspondiente, la cual en realidad no está incluida en esta etapa de la simulación. Se observa que la tasa de dosis queda por debajo de 3 µSv/h dentro de la pared con un margen considerable, por lo que, si bien resta evaluar la dosis resultante de la suma de todas las contribuciones, se espera que el blindaje del búnker de guías sea adecuado. Se produjeron 10^{10} partículas en la simulación de fotones de fuente y 10^8 en la de fotones de activación. En la de neutrones y fotones *prompt*, para alcanzar la convergencia deseada, se registraron nuevas fuentes de distribuciones sobre una superficie a 10 cm más de profundidad en la pared, en una corrida de 10^9 partículas. Finalmente, se usaron dichas fuentes en una nueva simulación con 10^{10} producciones.

En esta etapa también se registraron *tallies* en el corte transversal del blindaje exterior empleado en la subsección anterior. Se obtuvieron dosis mucho menores a las registradas al utilizar los escapes del tramo E como fuente, por lo que no se espera que se modifiquen los resultados ya obtenidos en el blindaje exterior al sumarle la contribución proveniente del interior del búnker.

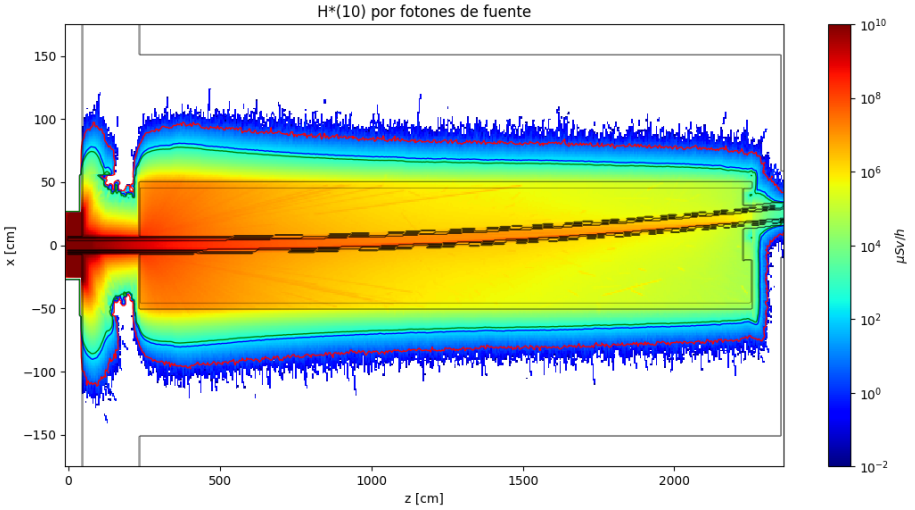
Por último, en la Figura 3.38 se comparan las cuatro componentes de dosis a lo largo del eje x en la región más comprometida del búnker de guías.



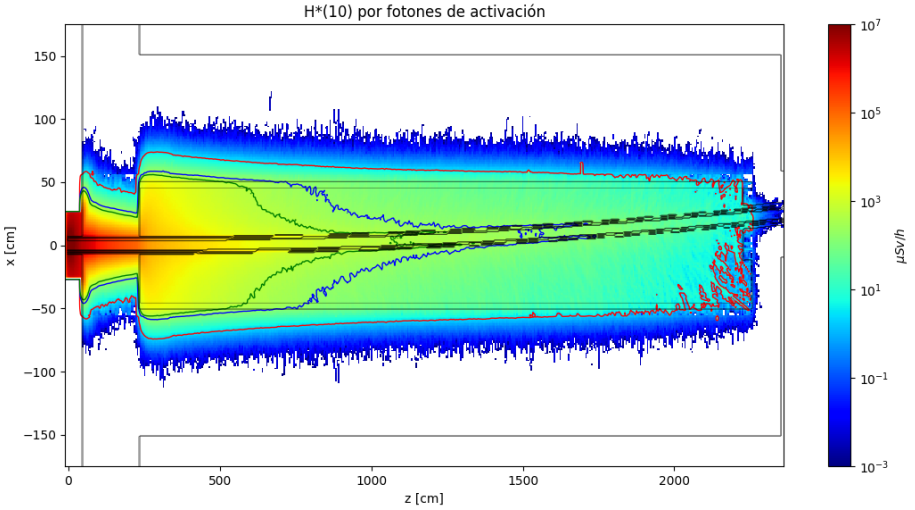
(a) Neutrones.



(b) Fotones *prompt*.

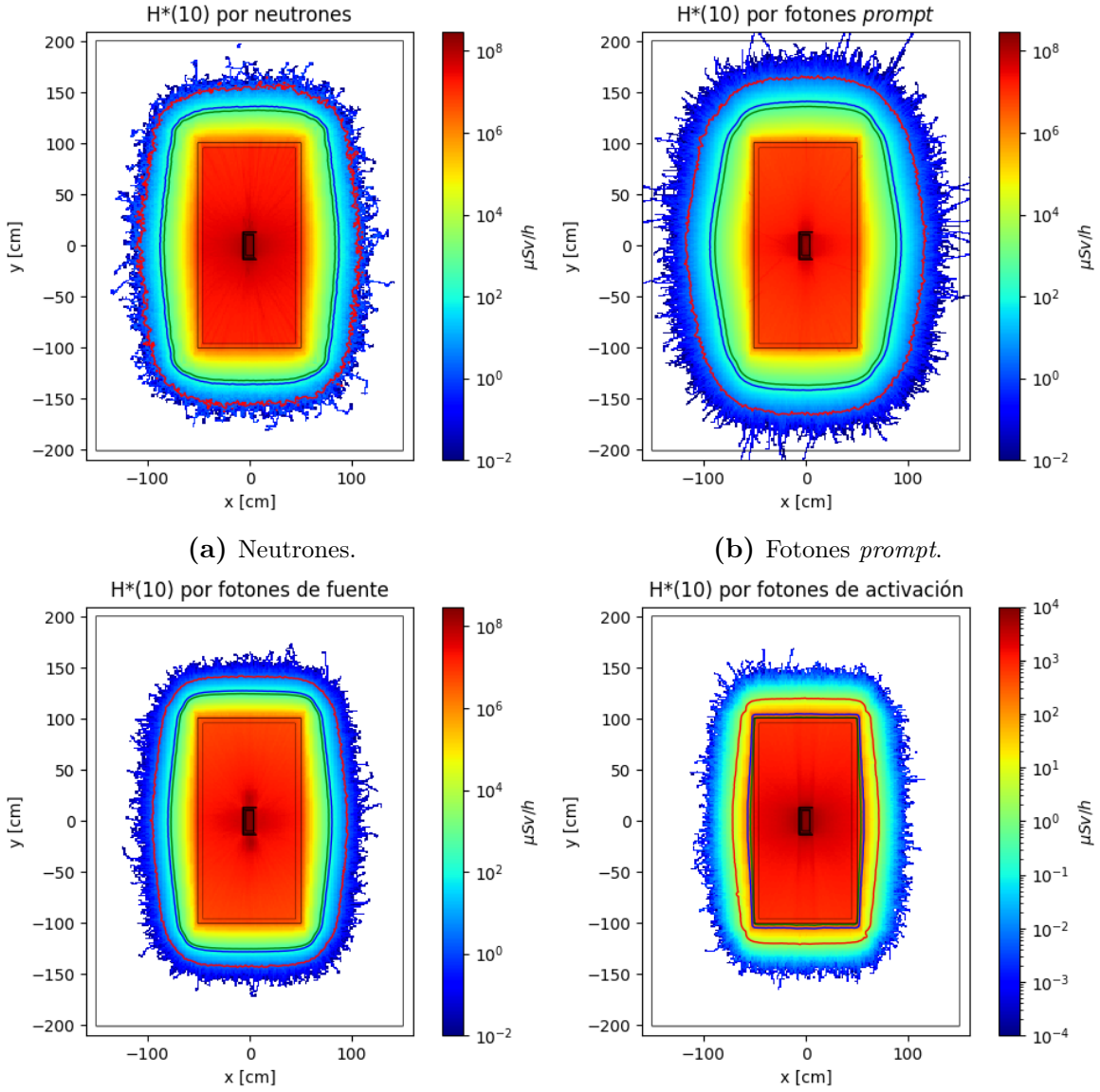


(c) Fotones de fuente.



(d) Fotones de activación.

Figura 3.36: Dosis ambiental en un corte horizontal del búnker de guías. En rojo la curva de 3 µSv/h, en azul la de 200 µSv/h y en verde la de 500 µSv/h.



(a) Neutrones.

(b) Fotones *prompt*.

(c) Fotones de fuente.

(d) Fotones de activación.

Figura 3.37: Dosis ambiental en un corte transversal del búnker de guías. En rojo la curva de 3 µSv/h, en azul la de 200 µSv/h y en verde la de 500 µSv/h.

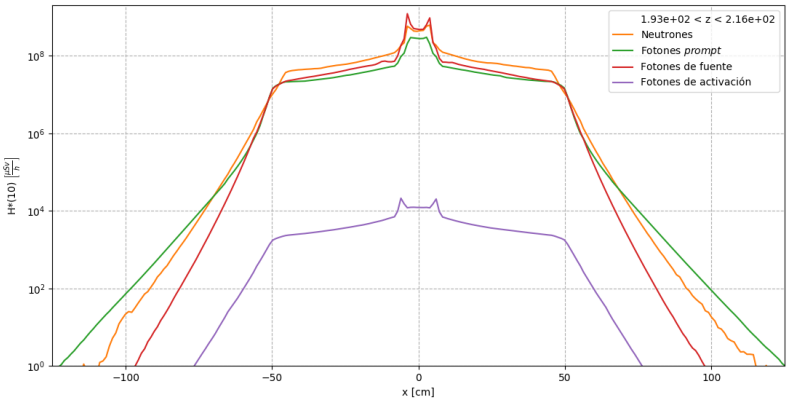


Figura 3.38: Comparación entre las componentes de dosis en un corte transversal del búnker de guías.

3.5.6. Resultados finales

En la Tabla 3.4 se muestra un resumen de las simulaciones realizadas, así como los recursos computacionales involucrados en cada etapa.

Zona	Part.	Programa	Proc.	Part. Gen.	T. cómp. [hs]
Tubo de vuelo	n	McStas	1	2.99E+06	0.001
Tubo de vuelo	γ_{fuente}	McStas	1	6.86E+05	0.001
Guía	n	McStas	1	1.00E+10	15.84
Guía	γ_{fuente}	McStas	1	1.00E+10	10.77
Búnker	n, γ_{prompt}	Tripoli	8	1.00E+09	27.31
Búnker	γ_{fuente}	Tripoli	16	1.00E+09	22.69
Búnker	γ_{activ}	Tripoli	8	1.00E+09	5.42
Blindaje búnker	n, γ_{prompt}	Tripoli	8	1.00E+09	6.30
Blindaje búnker 2	n, γ_{prompt}	Tripoli	16	1.00E+10	71.12
Blindaje búnker	γ_{fuente}	Tripoli	16	1.00E+10	22.89
Blindaje búnker	γ_{activ}	Tripoli	1	1.00E+08	2.51
Blindaje exterior	n, γ_{prompt}	Tripoli	8	1.00E+10	91.23
Blindaje exterior	γ_{activ}	Tripoli	1	1.00E+07	0.48
Total					276.57

Tabla 3.4: Esquema de las simulaciones realizadas. Para cada simulación se muestra la zona, partículas, programa, cantidad de procesadores utilizados, cantidad de partículas generadas y tiempo de cálculo.

En las siguientes imágenes se muestran los resultados obtenidos de la suma de todas las componentes de dosis. En las Figuras 3.39 y 3.40 se muestran los mapas de H*(10) en el búnker de guías, mientras que en las Figuras 3.41 y 3.42 se muestran aquellos obtenidos en el blindaje exterior. En todos los casos se observa que las curvas de nivel de 3 µSv/h se encuentran claramente dentro de los blindajes. Las regiones mapeadas fueron elegidas por ser aquellas más comprometidas, es decir donde, de no ser suficiente el blindaje, primero se observaría una dosis inaceptable del lado del *hall* de guías. De este modo se concluye que el diseño final de los blindajes del búnker, con 1 metro de hormigón pesado y 5 cm de parafina borada y el blindaje exterior, con 25 cm de hormigón pesado y 7 cm de plomo, resulta adecuado para verificar el requisito de una dosis equivalente ambiental menor a 3 µSv/h en el *hall* de guías. Por último, al haber realizado el presente cálculo de blindajes con recursos de cómputo y tiempos del proyecto limitados, se evidenció la gran utilidad de un método de reducción de varianza como la herramienta desarrollada.

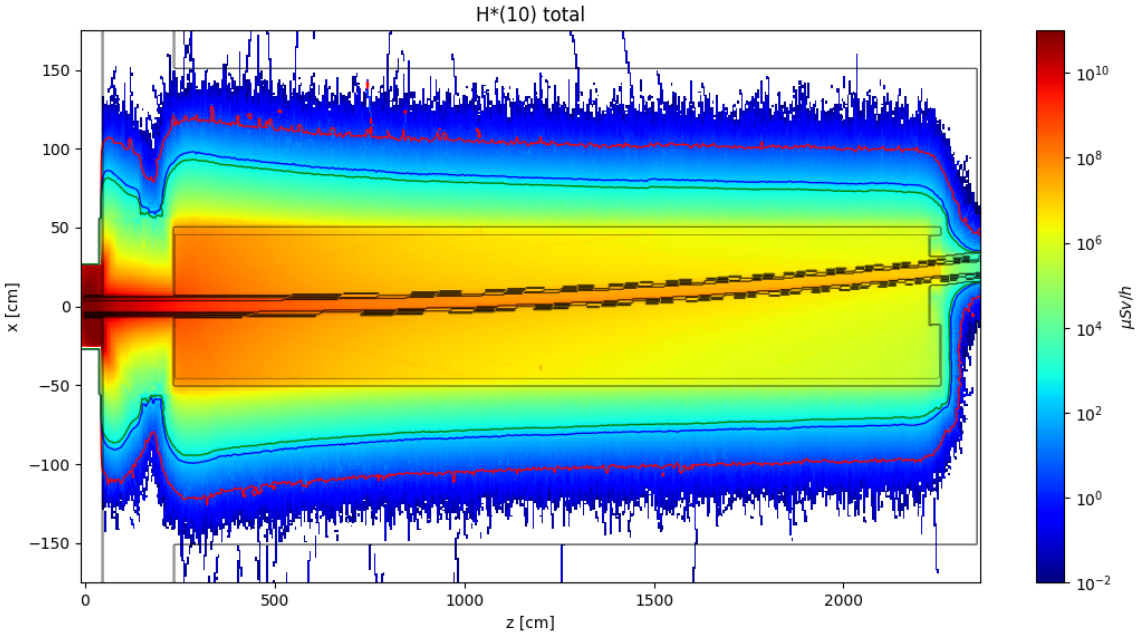


Figura 3.39: Dosis total en un plano horizontal del búnker, promediada entre y = -5 cm e y = 5 cm. En rojo la curva de 3 µSv/h, en azul la de 200 µSv/h y en verde la de 500 µSv/h.

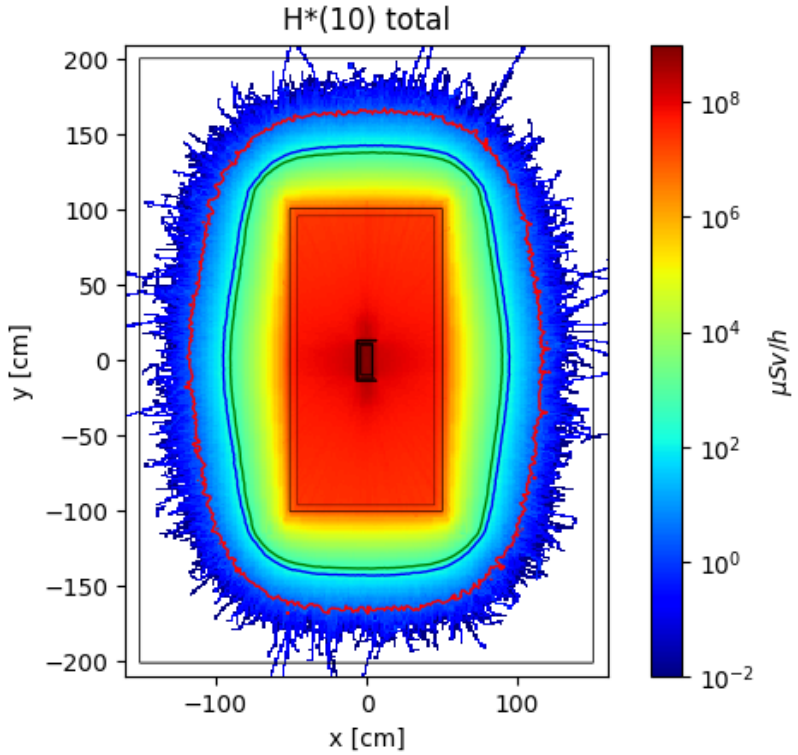


Figura 3.40: Dosis total en un plano transversal del búnker, promediada entre z = 350 cm y z = 450 cm. En rojo la curva de 3 µSv/h, en azul la de 200 µSv/h y en verde la de 500 µSv/h.

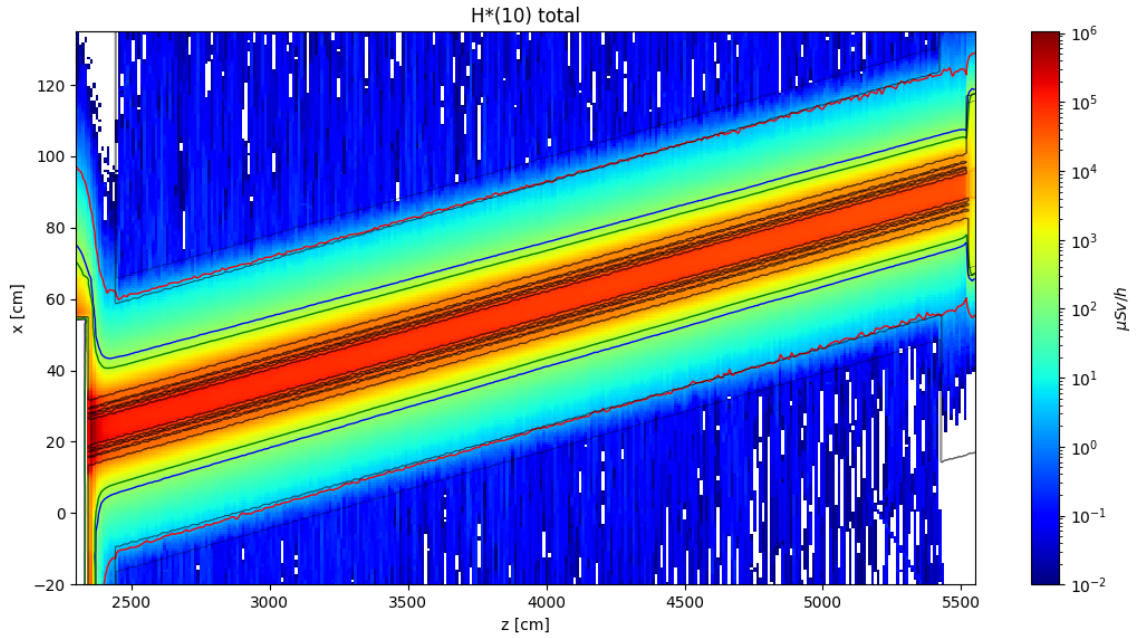


Figura 3.41: Dosis total en un plano horizontal del blindaje exterior, promediada entre y = -5 cm e y = 5 cm. En rojo la curva de 3 µSv/h, en azul la de 200 µSv/h y en verde la de 500 µSv/h.

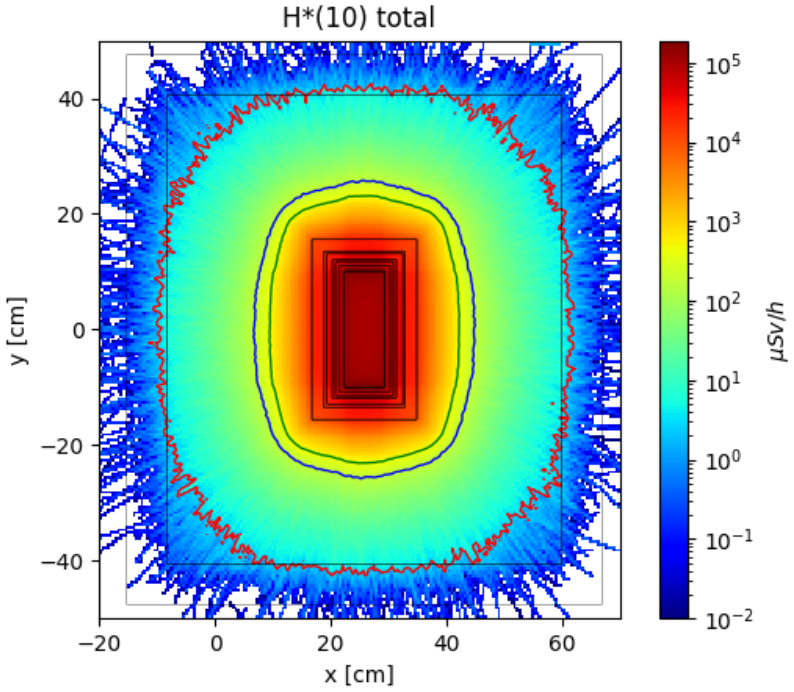


Figura 3.42: Dosis total en un plano transversal del blindaje exterior, promediada entre z = 2480 cm y z = 2520 cm. En rojo la curva de 3 µSv/h, en azul la de 200 µSv/h y en verde la de 500 µSv/h.

Capítulo 4

Conclusiones

En el presente proyecto integrador se construyó una biblioteca en lenguaje C, denominada `dsource`, para el modelado y uso de fuentes de distribuciones en códigos Monte Carlo. El esquema de las mismas se basó en los desarrollos realizados por Fairhurst y Ayala, pero la estructura computacional fue modificada sustancialmente. La librería implementada contiene las herramientas principales que constituyen una fuente de distribuciones independientemente de su geometría, estructura de discretizaciones y código en el cual se la pretende utilizar, de las cuales las principales son las funciones de guardado y sorteo. Mediante esta herramienta es posible caracterizar una corriente de partículas durante una simulación, registrándola en archivos de texto, para luego utilizarla como fuente en una nueva simulación, sin límites en la cantidad de partículas nuevas a generar.

Se implementaron, a modo de extensiones de la biblioteca `dsource`, tres paquetes de funciones para la utilización de tres geometrías particulares. Éstas fueron:

- Fuente plana rectangular.
- Fuente con forma de tubo de sección rectangular, para el modelado de escapes de radiación por los espejos de guías neutrónicas.
- Fuente volumétrica, para el modelado de fuentes de activación.

Estas sub-librerías permiten la utilización fluida de fuentes con las tres geometrías mencionadas, para los usos descritos anteriormente. En el primero de los casos se reprodujo el tipo de fuentes implementadas por Fairhurst y Ayala en sus proyectos integradores, mientras que las otras dos geometrías fueron diseñadas en el presente proyecto.

Los códigos Monte Carlo de interés en el presente proyecto son el McStas y el Tripoli, por lo que se construyeron los archivos necesarios para la comunicación de la biblioteca desarrollada con éstos. Esto permite el aprovechamiento de otra de las capacidades del desarrollo realizado, que es la conexión entre códigos. En el caso de McStas los detectores y fuentes se implementaron como componentes equivalentes a los incluidos por el

código. Por el otro lado, en Tripoli se utilizó la funcionalidad de fuente externa para las fuentes, y el registro de fuentes de *tracks* y posterior procesamiento para la detección. Uno de los desarrollos más notables en este caso fue la implementación de guía de neutrones en McStas, con registro de los neutrones no reflejados (escapes) en fuentes de distribuciones. Este componente, cuyo desarrollo es uno de los objetivos centrales del proyecto integrador, realiza simultáneamente la propagación de los neutrones a lo largo de una guía y el guardado de aquellos que no son reflejados en una fuente de distribuciones de tipo guía.

Considerando que las guías de neutrones son posibles de simular en McStas pero no en Tripoli, y por el otro lado, este último permite el cálculo de blindajes, lo cual no es posible en McStas, la combinación de ambos con la herramienta `dsource` permite realizar cálculos de blindajes asociados a guías de neutrones, completando el objetivo final del proyecto integrador. En este sentido, se realizó un diseño conceptual de blindaje para la guía del haz GF1 del RA10, incluyendo un búnker de guías y un blindaje rodeando el último tramo de la guía, por fuera del búnker, para el cual se verificó el cumplimiento de la restricción de dosis equivalente ambiental de 3 µSv/h fuera del búnker. La cadena de cálculo incluyó 4 simulaciones en McStas y 9 en Tripoli. Durante las mismas se utilizaron los desarrollos de fuentes de distribuciones tanto en la entrada y caras de la guía como en las paredes del búnker, así como para las fuentes volumétricas de fotones de activación. Esta secuencia de múltiples utilizaciones de fuentes de distribuciones permitió propagar las diferentes fuentes de radiación hacia el exterior de los blindajes, con tiempos de cálculo compatibles con los tiempos del presente proyecto.

Como trabajos futuros se propone la consolidación de una herramienta similar a la biblioteca `dsource`, para la implementación general de fuentes de distribuciones. Sería conveniente, en lugar de histogramas discretos, emplear técnicas más avanzadas de estimación de densidad. Por otra parte, sería de gran utilidad contar con mecanismos automatizados para lo obtención de discretizaciones de las variables, o su equivalente en la técnica a utilizar.

Por último, también se deja para trabajos posteriores una implementación más realista del búnker y *hall* de guías del RA10, con sus materiales y geometrías reales.

Apéndice A

Documentación de la biblioteca DSource

A.1. Estructuras

Definición de las estructuras que componen el paquete `dsource`.

A.1.1. DSource

```
typedef struct DSource{
    char *name;
    int ipt;
    Distrib **distribs;
    double **doms;
    int vlen;
    void (*transf0)(double*, double*, double*, int, double*);
    double trasl[3];
    double rot[3];
    double *gpar;
    int gpar_len;
} DSource;
```

Modela la fuente de distribuciones en su conjunto.

Parámetros:

- `char *name`: Nombre de la fuente de distribuciones. Ej.: `"entrada"`, `"guia_D"`, `"activ_fe"`.
- `int ipt`: Tipo de partícula. 1 para neutrón, 2 para fotón.
- `Distrib **distribs`: Lista de los árboles de distribuciones que componen la fuente. Longitud: `vlen`.

- `double **doms`: Lista de intervalos de dominio de cada variable, de la forma `[vmin, vmax]`. Longitud: `vlen`.
- `int vlen`: Longitud del vector de parametrización (véase 2.4.1).
- `void (*transf0)(double *, double *, double *, int, double *)`: Transformación de parametrización relativa, entre vector de parametrización y coordenadas relativas (definición en A.3).
- `double trasl[3]`: Vector de traslación.
- `double rot[3]`: Vector de rotación, en notación axial-angular, en radianes.
- `double *gpar`: Vector de parámetros geométricos. Longitud: `gpar_len`.
- `int gpar_len`: Longitud de `gpar`.

A.1.2. Distrib

```
typedef struct Distrib{
    char *name;
    int iv;
    Grid *grid;
    double *I;
    double *p2;
    int *N;
    double *cdf;
    struct Distrib *prev;
    struct Distrib **nexts;
} Distrib;
```

Modela la distribución de intensidad en función de una variable.

Parámetros:

- `char *name`: Nombre de la variable. Ej.: `"x"`, `"ekin"`.
- `int iv`: Índice de variable. Posición de la variable representada en el vector de parametrización, entre `0` y `vlen-1`.
- `Grid *grid`: Grilla con las discretizaciones a utilizar.
- `double *I`: Histograma de intensidad, es decir suma de pesos estadísticos. Longitud: `grid->nb`.
- `double *p2`: Histograma de suma de pesos cuadrados. Cada valor representa el cuadrado del error estadístico de la intensidad respectiva. Longitud: `grid->nb`.
- `int *N`: Histograma de cuentas, es decir cantidad de partículas registradas. Longitud: `grid->nb`.

- `double *cdf`: Función acumulativa de densidad. Longitud: `grid->nb`.
- `struct Distrib *prev`: Puntero a la distribución madre. `NULL` en caso de no poseerla.
- `struct Distrib **nexts`: Lista de punteros a las distribuciones hijas. Longitud: `grid->nb`. `NULL` en caso de no poseerlas.

A.1.3. Grid

```
typedef struct Grid{
    double *bins;
    int nb;
    double dom[2];
    char *form;
    char *units;
} Grid;
```

Modela la discretización de una variable.

Parámetros:

- `double *bins`: Posiciones de corte entre *bins*. Longitud: `vlen+1`. `NULL` en caso de que `form` sea `"lin"` o `"log"`.
- `int nb`: Cantidad de grupos en los que se discretiza la variable.
- `double dom[2]`: Límite inferior y superior del intervalo de dominio de la variable.
- `char *form`: Formato de la discretización, pudiendo ser `"tab"`, `"lin"` o `"log"`. Si es `"tab"` los límites entre *bins* son los listados en `bins`, si es `"lin"` son equiespaciados, y si es `"log"` sus logaritmos están equiespaciados.
- `char *units`: Unidades de la variable. Ej.: `"cm"`, `"MeV"`, `"rad"`.

A.2. Funciones de interfaz de usuario

A continuación se definen las funciones que operan sobre la estructura `DSource`, mediante las cuales se utilizan las fuentes de distribuciones.

A.2.1. DS_create

```
DSource *DS_create(char *name, int ipt, int vlen, Distrib **distribs,
   double **doms, Transf transf0, double *trasl, double *rot, double *
   gpar, int gpar_len)
```

Aloca una estructura `DSource` e inicializa sus parámetros con los valores provistos.

Argumentos: Los significados de todos los argumentos son los mismos que los de los parámetros de la estructura `DSource`.

Valor de retorno: Puntero a una estructura `DSource` alocada, cuyos parámetros poseen los valores provistos en los argumentos.

A.2.2. DS_I_tot

```
double DS_I_tot(DSource *dsource)
```

Calcula la intensidad total de la fuente, es decir la suma de los pesos de todas las partículas guardadas.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones cuya intensidad total se busca calcular.

Valor de retorno: Valor de intensidad total.

A.2.3. DS_p2_tot

```
double DS_p2_tot(DSource *dsource)
```

Calcula la suma de los pesos al cuadrado de todas las partículas guardadas.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones cuya suma de pesos cuadrados se busca calcular.

Valor de retorno: Valor de suma de pesos cuadrados.

A.2.4. DS_N_tot

```
int DS_N_tot(DSource *dsource)
```

Calcula la cantidad total de partículas guardadas.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones cuya cantidad de partículas se busca calcular.

Valor de retorno: Valor de cantidad de partículas.

A.2.5. DS_transf

```
void DS_transf(DSource *dsource, double *coord, double *vect, double *p
   , int inverse)
```

Función de parametrización absoluta, entre vector de parametrización y coordenadas absolutas. Convierte en una u otra dirección en función del valor de `inverse`. Según el caso los argumentos `coord`, `vect` y `p` pueden servir para proveer información o para recibir el resultado de la transformación.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones de la cual obtener la función de parametrización relativa, y los vectores de posición y traslación.
- `double *coord`: Coordenadas absolutas.
- `double *vect`: Vector de parametrización.
- `double *p`: Peso estadístico.
- `int inverse`: Indica la dirección de la transformación: de coordenadas absolutas a vector de parametrización si `inverse=0` y viceversa en caso contrario.

A.2.6. DS_is_in

```
int DS_is_in(DSource *dsource, double *coord)
```

Indica si una partícula pertenece a todos los dominios de una fuente de distribuciones.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones de donde obtener los dominios.
- `double *coord`: Coordenadas absolutas de la partícula.

Valor de retorno: 1 si la partícula pertenece a todos los dominios y 0 en caso contrario.

A.2.7. DS_save

```
int DS_save(DSource *dsource, double *coord)
```

Guarda una partícula en las distribuciones de una fuente de distribuciones.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones donde guardar la partícula.
- `double *coord`: Coordenadas absolutas de la partícula.

Valor de retorno: 0 en caso de realizar un guardado exitoso y 1 en caso contrario.

A.2.8. DS_sort

```
int DS_sort(DSource *dsource, double *coord)
```

Sortea una partícula en base a las distribuciones de una fuente de distribuciones.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones de donde obtener las distribuciones.
- `double *coord`: Vector donde guardar las coordenadas absolutas de la partículas sorteada.

Valor de retorno: 0 en caso de realizar un sorteo exitoso y 1 en caso contrario.

A.2.9. DS_save_distribs

```
void DS_save_distribs(DSource *dsource, char *folder, char **(*HFun)(
    DSource *dsource, char **headers))
```

Crea un directorio y guarda allí las distribuciones de una fuente de distribuciones en archivos de texto.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones cuyas distribuciones se busca guardar.
- `char *folder`: Nombre del directorio donde guardar las distribuciones. Si no existe se lo crea.
- `char **(*HFun)(DSource *dsource, char **headers)`: Función de encabezamientos. Genera la lista de *string* a guardar en la primera línea de cada archivo.

A.2.10. DS_read_info

```
void DS_read_info(DSource *dsource, char *file_info, int gpar_len)
```

Lee de un archivo de información (véase 2.4.1) los valores de `ipt`, `trasl`, `rot` y `gpar`, y los guarda en una fuente de distribuciones.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones donde guardar la información leída.
- `char *file_info`: Nombre del archivo de información.
- `int gpar_len`: Valor del parámetro `gpar_len`.

A.2.11. DS_read_tracks

```
void DS_read_tracks(DSource *dsource, char *file, char *MCcode, int N_
    part)
```

Lee una lista de *tracks* con formato de Tripoli o MCNP (PTRAC) y guarda las partículas en una fuente de distribuciones.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones donde guardar las partículas.
- `char *file`: Nombre del archivo con la lista de *tracks*.
- `char *MCcode`: Código Monte Carlo. Debe ser `"Tripoli"` o `"MCNP"`.
- `int N_part`: Máxima cantidad de partículas a leer. De valer 0 se leen todas las partículas en la lista.

A.2.12. DS_destroy

```
void DS_destroy(DSource *dsource)
```

Destruye una fuente de distribuciones, liberando toda la memoria asociada. Todas las estructuras `Grid` contenidas en la fuente deben estar alocadas en sitios diferentes, de lo contrario se incurrirá en un error de doble liberación de memoria. En general es preferible usar la función de destrucción específica para cada geometría.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones a destruir.

A.3. Funciones de geometría

A continuación se describen las funciones que deben definirse para definir una geometría, junto con su estructura de discretizaciones. En los nombres de las funciones debe remplazarse `[geom]` por el nombre de la geometría. En el presente proyecto éstas fueron definidas para las geometrías `window`, `guide` y `activ`.

A.3.1. [geom]_transf

```
void [geom]_transf(double *coord, double *vect, double *p, int inverse,
    double *gpar)
```

Transformación de parametrización relativa, entre coordenadas relativas y vector de parametrización. Convierte en una u otra dirección en función del valor de `inverse`. Según el caso los argumentos `coord`, `vect` y `p` pueden servir para proveer información o para recibir el resultado de la transformación.

Argumentos:

- `double *coord`: Coordenadas relativas.
- `double *vect`: Vector de parametrización.
- `double *p`: Peso estadístico.
- `int inverse`: Indica la dirección de la transformación: de coordenadas relativas a vector de parametrización si `inverse=0` y viceversa en caso contrario.
- `double *gpar`: Parámetros geométricos de la fuente de distribuciones.

A.3.2. [geom]_from_grids

```
DSource *[geom]_from_grids(char *folder, char *name, int ipt, double *
   trasl, double *rot, double *gpar)
```

Aloca y crea una fuente de distribuciones de geometría `geom` vacía, en base a un directorio con grillas con las discretizaciones de variables.

Argumentos:

- `char *folder`: Nombre del directorio donde encontrar los archivos de discretizaciones.
- `char *name`: Nombre de la fuente de distribuciones.
- `int ipt`: Tipo de partícula. 1 para neutrón, 2 para fotón.
- `double *trasl`: Vector de traslación.
- `double *rot`: Vector de rotación, con notación axial-angular, en radianes.
- `double *gpar`: Parámetros geométricos.

Valor de retorno: Puntero a la fuente de distribuciones creada e inicializada con las grillas y demás parámetros provistos.

Para las geometrías definidas, los archivos que debe contener el directorio son los siguientes:

- `window`: `UEmacro.txt`, `Emacro.txt`, `Emicro.txt`, `Xmacro.txt`, `Xmicro.txt`, `Ymacro.txt`, `Ymicro.txt`, `Umacro.txt`, `Umicro.txt` y `Pmicro.txt`. Donde `UE` se refiere a la grilla de μ que precede a la micro energía, `E` se refiere a la energía, `U` se refiere a μ y `P` se refiere a ϕ.
- `guide`: `Zmacro.txt`, `Zmicro.txt`, `Mmacro.txt`, `Mmicro.txt`, `Emacro.txt`, `Emicro.txt`, `Umacro.txt`, `Umicro.txt` y `Pmicro.txt`. Donde `M` se refiere a la variable `mirror`.

- `activ`: `E.txt`, `Z.txt`, `X.txt` y `Y.txt`. Nótese que en este tipo de fuentes no se emplean grillas macro y micro, ni grillas para μ y ϕ pues la producción es isotrópica.

Los archivos con las discretizaciones, para las geometrías implementadas, deben cumplir alguno de los siguientes formatos. Cada parámetro de la estructura `Grid` debe remplazarse por el valor deseado.

- Grilla equiespaciada linealmente:

```
nb dom[0] dom[1]
```

- Grilla tabulada:

```
bins[0]
bins[1]
...
bins[nb-1]
```

- Conjunto de grillas equiespaciadas para `x` ó `y` en función de macro `E`, exclusivo para la geometría `window`:

```
dom[0] dom[1]
nb_0
nb_1
...
nb[Emacro.nb-1]
```

A.3.3. [geom]_from_distribs

```
DSource *[geom]_from_distribs(char *folder, char *name, int sort)
```

Aloca y crea una fuente de distribuciones, y lee las distribuciones y discretizaciones de un conjunto de archivos de distribuciones, creado por `DS_save_distribs`. Para ello hace uso de la información auxiliar provista en la primera línea de cada archivo, generada por `[geom]_headers`.

Argumentos:

- `char *folder`: Nombre del directorio con el conjunto de archivos de distribuciones.
- `char *name`: Nombre de la grilla a crear.

- `int sort`: Si vale 1, antes de guardar cada valor de intensidad I se le suma un número aleatorio con distribución gaussiana y desviación estándar igual a $\sqrt{p2}$, es decir el error estadístico. Si vale 0 se guardan los valores de I sin modificación.

Valor de retorno: Puntero a la fuente de distribuciones creada, con las distribuciones cargadas.

A.3.4. [geom]_headers

```
char **[geom]_headers(DSource *dsource, char **headers)
```

Genera un conjunto de *strings* para guardar en la primera línea de cada archivo de distribuciones. Debe contener la información necesaria para leer las distribuciones, lo cual incluye número de grupos de cada grilla y límite superior de cada dominio. Su estructura es libre, pero debe ser coherente con la interpretación que realice `[geom]_from_distribs`.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones de donde obtener la información a guardar.
- `char **headers`: Lista de *strings* alocados pero vacíos. Longitud: `vlen`.

Valor de retorno: `headers`.

A.3.5. [geom]_from_tally

```
DSource *[geom]_from_tally(char *tally, const char *score, char *
   spectrum, char *name, int sort, int ipt, double *trasl, double *rot)
```

Aloca y crea una fuente de distribuciones, y lee la distribución registrada en un *tally* de Tripoli. Se implementó únicamente para la geometría `activ`.

Argumentos:

- `char *tally`: Nombre del archivo de *output* de Tripoli. Puede contener más *tallies* (*scores*) que el que se busca leer.
- `const char *score`: Nombre del *score* de Tripoli a leer.
- `char *spectrum`: Archivo de espectro energético, con el formato de los archivos CSV descargados de la base de datos de IAEA [6].
- `char *name`: Nombre de la fuente de distribuciones a crear.
- `int sort`: Si vale 1, antes de guardar cada valor de intensidad se le suma un número aleatorio con distribución gaussiana y desviación estándar igual al error provisto por Tripoli, es decir el error estadístico. Si vale 0 se guardan los valores de intensidad sin modificación.

- `int ipt`: Tipo de partícula. 1 para neutrón, 2 para fotón.
- `double *trasl`: Vector de traslación.
- `double *rot`: Vector de rotación, con formato axial-angular, en radianes.

Valor de retorno: Puntero a la fuente de distribuciones creada, con las distribuciones cargadas.

A.3.6. [geom]_destroy

```
void [geom]_destroy(DSource *dsource)
```

Destruye una fuente de distribuciones, liberando toda la memoria asociada.

Argumentos:

- `DSource *dsource`: Fuente de distribuciones a destruir.

A.4. Funciones para gráficos en Python

A continuación se describen las funciones para gráficos definidas en el paquete `dsource` en Python. Éste también incluye todas las funciones descritas anteriormente, con sintaxis y funcionamiento análogo.

A.4.1. Distrib.plot

```
Distrib.plot(self, ind=[], show=True, scale="lin", fact=1, errorbar=
   True, ylabel="J", label="", norm=False, steps=True)
```

Crea un gráfico de una distribución de densidad de corriente (intensidad dividido el ancho de cada *bin*) en función de una variable. Utiliza las funciones `plot` o `errorbar` de `matplotlib.pyplot`.

Argumentos:

- `ind`: Lista de índices enteros indicando la distribución a graficar, dentro del árbol de distribuciones. `[]` indica la distribución sobre la que fue llamada, y cada número en la lista indica el número de rama al cual llamar recursivamente.
- `show`: Indica si invocar `matplotlib.pyplot.show()` dentro de la función o no.
- `scale`: Escala de los ejes, pudiendo se `"lin"`, `"xlog"`, `"ylog"` o `"log"`.
- `fact`: Factor de multiplicación a aplicar sobre las intensidades graficadas, muy útil para convertir sus valores a unidades físicas. Cada vez que la función se llama recursivamente sobre las ramas indicadas en `ind` se divide `fact` por el ancho del *bin* correspondiente.

- `errorbar`: Indica si graficar las barras de error, o no.
- `ylabel`: Etiqueta del eje vertical.
- `label`: Leyenda del gráfico.
- `norm`: Indica si normalizar el gráfico, sobrescribiendo el valor de `fact`, o no.
- `steps`: Indica si graficar líneas horizontales sobre el ancho de cada *bin* (`True`) o si interpolar los valores entre los centros de cada *bin* (`False`).

Valor de retorno: `[vi, J, err]`, con `vi` la lista de límites entre *bins*, `J` la lista de densidades de corriente y `err` sus errores estadísticos.

A.4.2. Distrib.plot_I

```
Distrib.plot_I(self, ind=[], show=True, scale="lin", fact=1, ylabel="I
    ", errorbar=True, legend="")
```

Crea un gráfico de intensidad (suma de pesos estadísticos) en función de una variable. Utiliza las funciones `plot` o `errorbar` de `matplotlib.pyplot`.

Argumentos: Todos sus argumentos tienen los mismos significados que los de la función `Distrib.plot`.

Valor de retorno: `[vi, I, err]`, con `vi` la lista de límites entre *bins*, `I` la lista de intensidades y `err` sus errores estadísticos.

A.4.3. Distrib.plot_cdf

```
Distrib.plot_cdf(self, ind=[], show=True, scale="lin", ylabel="cdf",
    legend="")
```

Crea un gráfico de la función acumulativa de densidad, en función de una variable. Utiliza la función `plot` de `matplotlib.pyplot`.

Argumentos: Todos sus argumentos tienen los mismos significados que los de la función `Distrib.plot`.

Valor de retorno: `[vi, cdf]`, con `vi` la lista de límites entre *bins*, y `cdf` es la lista de valores de la función acumulativa de densidad.

A.4.4. Distrib.plot_2D

```
Distrib.plot_2D(self, ind=[], show=True, fact=1, log=False, title="J",
    norm=False)
```

Crea un gráfico 2D de densidad de corriente en función de dos variables. Utiliza la función `imshow` de `matplotlib.pyplot`, con `cmap='jet'`.

Argumentos:

- `ind`: Lista de índices enteros indicando la distribución a graficar, dentro del árbol de distribuciones. `[]` indica la distribución sobre la que fue llamada, y cada número en la lista indica el número de rama al cual llamar recursivamente.
- `show`: Indica si invocar `matplotlib.pyplot.show()` dentro de la función o no.
- `fact`: Factor de multiplicación a aplicar sobre las intensidades graficadas, muy útil para convertir sus valores a unidades físicas. Cada vez que la función se llama recursivamente sobre las ramas indicadas en `ind` se divide `fact` por el ancho del *bin* correspondiente.
- `log`: Indica si emplear una normalización logarítmica en la barra de colores.
- `title`: Título del gráfico.
- `norm`: Indica si normalizar el gráfico, sobrescribiendo el valor de `fact`, o no.

Valor de retorno: `[[vi_x, vi_y], J]`, con `vi_x` y `vi_x` las listas de límites entre *bins* de las dos variables y `J` la matriz de densidades de corriente.

Bibliografía

[1] Autoridad Regulatoria Nuclear. AR 4.1.1: Exposición ocupacional en reactores nucleares de investigación. 1, 35

[2] Fairhurst, R. E. Cálculo neutrónico detallado de haces y guías de neutrones del reactor RA-10. Proyecto Fin de Carrera, Instituto Balseiro, 6 2017. 2, 35, 36, 47, 51

[3] Ayala, J. E. Implementación de una línea de cálculo basada en el código Tripoli a problemas de blindaje del reactor RA-10. Proyecto Fin de Carrera, Instituto Balseiro, 6 2019. 2

[4] User and Programmers Guide to the Neutron Ray-Tracing Package McStas, version 2.5, 12 2018. 4

[5] TRIPOLI-4® Version 8 User Guide, 2 2013. 4

[6] Live chart of nuclides — IAEA Nuclear Data Services. URL https://www-nds.iaea.org/relnsd/vcharthtml/VChartHTML.html, accesed: 10-may-2020. 27, 86

[7] Aygün, B. Neutron and gamma radiation shielding properties of high-temperature-resistant heavy concretes including chromite and wolframite. *Journal of Radiation Research and Applied Sciences*, **12** (1), 352–359, 2019. URL https://doi.org/10.1080/16878507.2019.1672312. 39

Agradecimientos

Empezando por lo académico, agradezco a mis padres de PI Ariel y Nacho, que siempre estuvieron para orientarme.

Siguiendo por DeFRRa (Difra), estuvieron Alexis, Ana, Santiago, Jimmy, Edmundo, y varies otres respondiendo mis dudas computacionales con paciencia admirable, o simplemente haciendo las mañanas más agradables, con facturas, mates y muy buena onda.

En el CAB pero fuera de Difra agradezco a les amigues que hice que me acompañaron en estos tres años, desde mi grupo de tempraneros en las clases de Javier hasta mis compañeres nucleares con quienes tantas tardes compartí, pasando por mucha gente copada más.

Por último (y no menos importante), fuera del CAB agradezco a mi novia Anto, que me banca siempre en casa, y con la que aprendimos (aprendemos) a ser adultes. Agradezco a mi mamá, Teresa, y a mis hermanes Lu y Mai, a quienes, si bien no veo mucho, les mantengo siempre cerca. Agradezco a quienes me acompañaron en Buenos Aires, mi abuela, mis primes, Tomás y toda la gente copada de ese gran lugar llamado Exactas. Agradezco a la gente de Bolsón, Italia, y a todas las personas que me hicieron ser quien soy.

Aparte de las personas, agradezco a la educación pública sin la cual todo lo que hice no hubiera sido posible.